N° d'ordre :

THÈSE

présentée pour obtenir le titre de

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE

Par : Charles-Edmond BICHOT

École doctorale : Informatique et Télécommunications
Spécialité : Sûreté du Logiciel et Calcul à Haute Performance
Laboratoire d'accueil : Laboratoire d'Optimisation Globale (DSNA/ENAC)

ÉLABORATION D'UNE NOUVELLE MÉTAHEURISTIQUE POUR LE PARTITIONNEMENT DE GRAPHE : LA MÉTHODE DE FUSION-FISSION. APPLICATION AU DÉCOUPAGE DE L'ESPACE AÉRIEN

Soutenance le 9 novembre 2007 devant le jury composé de :

MM.	Patrick SIARRY	Université Paris XII	Rapporteur
	El-Ghazali TALBI	Université de Lille I	Rapporteur
MM.	François PELLEGRINI	ENSEIRB	Membre du jury
	Jean-Marc ALLIOT	DSNA	Membre du jury
MM.	Joseph NOAILLES	ENSEEIHT	Directeur de thèse
	Nicolas DURAND	DSNA	Codirecteur de thèse

Résumé : *Dans cette thèse, nous étudions des méthodes de partitionnement de graphe et les appliquons au découpage de l'espace aérien, ainsi qu'à d'autres problèmes. L'espace aérien est composé de volumes limités, appelés secteurs de contrôle, chacun étant sous la responsabilité d'un contrôleur. Chaque contrôleur est habilité sur un ensemble de secteurs, appelé zone de qualification. Les secteurs sont également regroupés en centres de contrôle, qui englobent au moins une zone de qualification. Dans le cadre du ciel unique européen, la Commission européenne a prévu la création de blocs fonctionnels d'espace aérien. La création de ces blocs entre pays européens entraînera probablement un redécoupage des centres actuels. Cette thèse propose des outils d'aide à la conception d'un nouveau découpage de l'espace européen en centres et en zones de qualification. À cet effet, plusieurs méthodes sont étudiées : des méthodes de partitionnement classiques, comme l'expansion de région, le multi-niveaux ou les algorithmes de type Kernighan-Lin ; des métaheuristiques, comme le recuit simulé, les algorithmes de colonies de fourmis et les algorithmes évolutionnaires ; et une nouvelle méthode que nous avons mise au point, la fusion-fission. C'est cette dernière qui permet de trouver les découpages les plus performants, au sens de la fonction de coût utilisée, pour le découpage de l'espace aérien. Afin de diversifier ses applications, nous l'avons aussi adaptée à la segmentation d'images et à la classification de documents. Enfin, la qualité de cette méthode a été éprouvée sur les bancs de tests classiques du partitionnement de graphe et confrontée aux méthodes concurrentes. Elle a permis de trouver pour plusieurs problèmes de test des partitions dont le coût est le plus bas obtenu jusqu'à présent.*

Mots clés : Graphe, partition, fusion-fission, contrôle aérien, métaheuristique, multi-niveaux, affinage, spectrale

---

Abstract : *This thesis studies graph partitioning methods and applies them to airspace partitioning and other partitioning problems. Each air traffic controller supervises a limited space, called an air traffic sector. Controllers have qualifications to work only on a set of sectors, called qualification air zone. Sectors are grouped together into control centers wich include almost one qualification air zone. The European single sky project intended by the European Commission could involve a new airspace partitioning into control centers and qualification air zones. In this framework, this thesis proposes some tools to design the airspace. Classical graph partitioning methods are studied (load-balancing, region growing and multilevel algorithms), a well as some metaheuristics (simulated annealing, ant colonies and evolutionary algorithms). A new method is introduced in this thesis : the fusion-fission method. Compared with the others, this method allows to find the best airspace partitioning for our objective function. To diversify its applications, the fusion-fission method has also been applied to image segmentation and documents clustering. Finally, it has been tested on classical benchmarks and compared with contestant methods. On benchmarks, it finds some new partitions which have the lowest cut ever found.*

Key words : Graph, partition, fusion-fission, airspace control, metaheuristic, multilevel, load balancing, spectral

LABORATOIRE D'OPTIMISATION GLOBALE
DIRECTION DES SERVICES DE LA NAVIGATION AÉRIENNE
ÉCOLE NATIONALE DE L'AVIATION CIVILE
7 avenue Édouard Belin - BP 54005
31055 Toulouse cedex 4

Table des matières

Introduction générale

Cette thèse a pour but d'apporter des méthodes et des outils pour le redécoupage de l'espace aérien européen. Elle s'est effectuée sous la direction de Nicolas Durand, au Laboratoire d'optimisation globale de la Direction des services de la navigation aérienne (DSNA) et de l'École nationale de l'aviation civile (ENAC). Le découpage de l'espace aérien se modélise sous la forme d'un problème de partitionnement de graphe. Nous avons donc cherché à étudier ce domaine qui a peu été abordé dans notre laboratoire. Ainsi, le sujet s'est progressivement orienté vers l'étude et la recherche de méthodes de partitionnement. Le découpage de l'espace aérien européen a cependant toujours été notre problématique centrale.

Plusieurs méthodes sont présentées dans cette thèse : des méthodes de partitionnement classiques, comme l'expansion de région, le multi-niveaux ou les algorithmes de type Kernighan-Lin ; des métaheuristiques, comme le recuit simulé, les algorithmes de colonies de fourmis et les algorithmes évolutionnaires ; et une nouvelle méthode que nous avons mise au point, la fusion-fission. Comme cette dernière a permis de trouver les découpages de l'espace aérien les plus performants, au sens de la fonction de coût utilisée, nous avons voulu diversifier ses applications. Ainsi, elle a été adaptée à la segmentation d'images et à la classification de documents. Afin de tester la qualité de cette méthode, nous l'avons aussi éprouvée sur les bancs de tests classiques du partitionnement de graphe.

La démarche que nous avons suivie au cours de cette thèse nous a influencés dans l'organisation de ce document. Ainsi, nous étudierons dans un premier temps les méthodes de partitionnement, puis dans un second temps nous présenterons leur application aux problèmes du trafic aérien, de la segmentation d'image, de la classification de documents et des bancs de tests classiques.

Le partitionnement

Le mot « partitionnement » est un néologisme correspondant au mot anglais *partitioning*, qui désigne l'action de créer une partition. La partition est le résultat de la division en parties d'un ensemble. En mathématiques, une partition d'un ensemble E est une famille de parties de E, disjointes deux à deux et dont la réunion est l'ensemble E. Tout élément de E appartient donc à une et une seule de ces parties.

Le partitionnement est utilisé pour résoudre un grand nombre de problèmes d'ingénierie. Les exemples d'applications du partitionnement sont multiples : l'exploration de données, la conception de circuits intégrés électroniques, la répartition de charge pour les machines parallèles, la dynamique des fluides, le calcul matriciel, le trafic aérien, etc.

Créer une partition consiste à répartir un ensemble d'objets en plusieurs sous-ensembles. Dans le but de répartir les objets dans ces différents sous-ensembles, il peut être utile de pouvoir les comparer entre eux. Ainsi, chaque objet va être associé à d'autres objets, et les associations résultantes, ou liens, devront pouvoir être quantifiées. De plus, si l'ensemble des ces objets est fini, ou au moins dénombrable, alors toutes les conditions sont réunies pour créer un graphe de ces objets, c'est-à-dire concrétiser les associations entre objets. Il ne reste plus qu'à construire une partition de ce graphe. L'objectif du partitionnement d'un ensemble d'objets est en général de répartir ceux-ci en parties ayant de très forts liens internes et de faibles liens entre les parties. C'est ce que l'on appelle l'objectif du partitionnement ou fonction objectif. Cette fonction objectif varie selon le problème précis à résoudre.

Le problème du partitionnement de graphe est commun à plusieurs disciplines :

- il fait partie des problèmes de théorie des graphes et par là même des mathématiques discrètes. Les mathématiques discrètes sont l'étude des structures discrètes, c'est-à-dire des ensembles finis ou dénombrables ;
- il appartient aussi aux problèmes d'optimisation combinatoire. Un problème d'optimisation combinatoire consiste à trouver la meilleure solution au sein d'un ensemble discret de solutions ;
- il est résolu grâce aux outils informatiques.

Le partitionnement de graphe se situe donc à cheval entre l'informatique et les mathématiques appliquées. Les mathématiques discrètes et l'optimisation combinatoire sont liées à la recherche opérationnelle. La recherche opérationnelle est une discipline transversale qui regroupe les mathématiques appliquées, l'informatique et les sciences de l'ingénieur. Son but est de résoudre des problèmes de décision complexes issus du monde réel, en particulier des problèmes d'optimisation. Le partitionnement de graphe est un problème de recherche opérationnelle.

Notre contribution

Notre contribution à l'avancement des connaissances nous paraît s'organiser suivant 4 axes :

- cette thèse présente un **état de l'art** du partitionnement de graphe. Cette spécialité n'étant pas présente dans notre laboratoire, il nous a fallu en premier identifier parmi les différentes méthodes de partitionnement, celles qui concernaient les graphes. Puis nous avons classifié les méthodes restantes en plusieurs catégories : multi-niveaux, spectrales, expansion de région, affinage, *etc*. Enfin, la présentation de cet état de l'art est le fruit d'une analyse critique de ces méthodes. En plus de l'état de l'art du partitionnement de graphe, cette thèse comporte une présentation détaillée de trois métaheuristiques : le recuit simulé, les algorithme de colonies de fourmis et les algorithmes évolutionnaires ;
- plusieurs **innovations** sont apportées par cette thèse. La principale est la création de la méthode de fusion-fission et des deux algorithmes de partitionnement de graphe qui en découlent. Cependant, d'autres nouveaux algorithmes pour le partitionnement de graphe sont aussi proposés : l'algorithme basé sur la percolation, l'algorithme de répartition de charge utilisant la notion de gain de Kernighan-Lin, les deux algorithmes de recuit simulé pour le partitionnement, l'algorithme de colonies de fourmis pour le partitionnement, ainsi que les trois algorithmes évolutionnaires pour le partitionnement ;
- nous avons appliqué le partitionnement de graphe à de nombreux problèmes. Parmi ces **applications**, la principale est le problème du découpage de l'espace aérien européen. Si cette dernière a nécessité l'étape de modélisation la plus complexe et la plus longue, toutes ces applications ont demandé une phase de modélisation importante. Puis a succédé à la modélisation de ces problèmes, l'étape d'adaptation des méthodes de partitionnement ;
- enfin, des nombreuses **évaluations** sont présentées dans cette thèse. Celles-ci permettent de comparer la performance des méthodes, notamment grâce à l'utilisation de bancs de tests classiques ; d'évaluer l'adaptation des méthodes au partitionnement de graphe, en particulier pour

les métaheuristiques ; d'éprouver les modélisations des problèmes traités ; et de vérifier la performance des nouvelles méthodes proposées.

Ce document s'organise en fonction de ces 4 axes. Une première partie étudie l'état de l'art du domaine du partitionnement de graphe et les innovations que nous avons apportées. La seconde partie présente les applications de cette thèse et les évaluations qui en découlent.

Les applications du partitionnement

Le découpage de l'espace aérien européen

Dans les régions de l'espace aérien où le contrôle des aéronefs est assuré, l'espace aérien est divisé en volumes limités placés sous la responsabilité des contrôleurs aériens. Ces volumes d'espace sont appelés *secteurs de contrôle*. Chaque contrôleur est habilité sur un ensemble de secteurs appelé *zone de qualification*. Les secteurs sont également regroupés en centres de contrôle qui englobent au moins une zone de qualification. Dans le cadre du projet de ciel unique européen, la Commission européenne a prévu la création de blocs fonctionnels d'espace aérien, en anglais *Fonctional Airspace Block* (FAB). La création de ces blocs entre pays européens entraînera probablement un redécoupage des centres actuels et *a fortiori* des zones de qualification.

Le problème du découpage de l'espace aérien s'appuie sur la configuration actuelle des secteurs de contrôle et des routes aériennes. Le but de ce découpage est d'augmenter la sûreté comme la fluidité du contrôle aérien. Afin de répondre aux exigences de la Commission européenne concernant la création des blocs fonctionnels, ce redécoupage doit aussi pouvoir faire abstraction des frontières nationales.

Notre étude consiste à former des centres de contrôle et des zones de qualification à partir de la connaissance des flux d'avions entre les secteurs. La première application du redécoupage concerne le ciel aérien français, mais est étendue au projet du FAB d'Europe centrale qui regroupe 6 pays européens : l'Allemagne, les Pays-Bas, la Belgique, le Luxembourg, la Suisse et la France. Afin d'avoir une vue globale de ce qu'un redécoupage de l'espace pourrait apporter en Europe, le partitionnement d'une région plus grande a aussi été étudié.

La *core area* est la région européenne ayant la plus forte densité de trafic. Celle-ci regroupe les espaces aériens de 11 pays : l'Espagne, l'Italie, la Suisse, l'Autriche, l'Allemagne, la Belgique, le Luxembourg, les Pays-Bas, le Danemark, la France et le Royaume-Uni. Cette région est constituée de 638 secteurs de contrôle, rassemblés en 31 centres et 50 zones de qualification. La partition du ciel à créer concerne donc un graphe de taille moyenne, mais pour un nombre de parties déjà significatif, compte tenu de cette taille. Ce redécoupage nécessite donc des outils de partitionnement de graphe performants.

Les données européennes qui sont en notre possession sont souvent incomplètes et possèdent de nombreuses incohérences. Ainsi, malgré tous nos efforts pour les rendre cohérentes, les données qui sont à la base de notre étude sont loin d'être parfaites. Cependant, les résultats présentés dans cette thèse concernant le redécoupage de l'espace aérien sont là pour évaluer la performance des algorithmes utilisés, et non pas pour proposer une solution directement applicable à l'échelle européenne.

Le problème du découpage de l'espace aérien qui est présenté dans cette thèse n'a encore jamais été étudié dans le Laboratoire d'optimisation globale. Le problème qui s'en rapproche le plus est celui de la sectorisation de l'espace, réalisé par Daniel Delahaye [DEL95]. Le problème de sectorisation consiste à découper l'espace en secteurs de contrôles. C'est un problème différent de notre sujet, car dans ce dernier les secteurs de contrôle ne sont pas modifiés, mais pris tels quels et répartis dans les zones de qualifications. Cependant, il pourrait certainement être très intéressant d'envisager de conjuguer les deux travaux afin de tester de nouveaux découpages possibles. Une autre étude du laboratoire, portant sur la construction de trajectoires 3D, réalisée par David Gianazza [GIA04], peut aussi être rapprochée de notre sujet. Dans une vision schématique, une première étude permettrait de construire un réseau de route, une seconde de créer les secteurs correspondant à ce réseau, et la dernière organiserait les zones de qualification en fonctions des résultats précédents.

Autres applications possibles

La conception de circuits intégrés électroniques

La conception de circuits intégrés électroniques est une tâche de plus en plus complexe, car le nombre de composants de ces circuits augmente régulièrement. Le problème du partitionnement d'un circuit intégré est de rassembler les différents éléments d'un circuit en sous-circuits, ou cellules, tels que le nombre de connexions entre ceux-ci soit minimal. Chaque sous-circuit peut alors être réalisé indépendamment, ce qui rend la conception et l'intégration du circuit plus rapide.

Ces cellules sont des modules qui remplissent des fonctions prédéfinies et qui possèdent des entrées et des sorties. Elles sont connectées entre elles au niveau de leurs entrées-sorties par une « toile » (*net* en anglais). Un circuit électronique est caractérisé par sa liste de cellules et sa liste de connexions (*netlist* en anglais).

Un graphe peut être utilisé pour représenter un circuit électronique, bien que la modélisation de ceux-ci sous forme d'hypergraphes soit plus courante [ALP95a, CON03]. Les cellules représentent les sommets du graphe, et les connexions ses arêtes. Les applications du partitionnement de graphe à la conception de circuits intégrés électroniques sont anciennes [KER70, WEI89].

La répartition de charge pour les machines parallèles

La répartition de charge pour les machines parallèles est un problème fondamental du calcul parallèle. Le but est de répartir correctement les charges de calcul entre les processeurs et de réduire autant que possible la durée des communications entre les processeurs.

La répartition de charge entre processeurs peut être dynamique ou statique. La répartition statique est utilisée pour répartir la charge de calcul à l'initialisation. La répartition dynamique répartit la charge durant la phase de calcul par réajustement progressif. La répartition dynamique est utilisée en plus de la répartition statique quand les variables du problèmes changent en cours d'exécution, ce qui provoque une modification de la charge de calcul.

Un graphe non-orienté permet de modéliser la répartition de charge pour les machines parallèles. Les sommets du graphe représentent des sous-blocs de calcul, souvent exécutés sur un unique processeur, et les arêtes, la communication nécessaire à la programmation parallèle.

Les problèmes d'ingénierie de grande taille, comme par exemple la résolution de systèmes linéaires ou la multiplication de matrices, nécessitent souvent l'utilisation d'une puissance de calcul bien plus importante que celle que peut fournir un ordinateur classique. La résolution de tels problèmes requiert un super-calculateur ou encore un *cluster* d'ordinateurs (réseau d'ordinateurs). Dans les deux cas, un outil de répartition de charge est nécessaire.

La segmentation d'images

La segmentation d'images permet d'isoler les différents objets présents sur une image. Une image se décompose en une matrice de pixels de couleurs ou de niveaux de gris. À partir de cette matrice il est possible de créer un graphe de l'image dont les sommets représentent les pixels et les arêtes la différence d'intensité lumineuse entre les pixels.

À partir de cette modélisation, la segmentation d'images se transforme en partitionnement de graphe. Cependant cette modélisation nécessite de trouver une fonction objectif pertinente à minimiser. Celle-ci doit permettre une segmentation visuellement acceptable de l'image. La littérature sur ce sujet est abondante [BEN05, FEL04, MAR04, WAN03, SHI00, WEI99], mais le partitionnement de graphe est loin d'être la seule méthode pour résoudre le problème de la segmentation d'images [SKA94].

La classification

Les outils de classification permettent de trier un ensemble d'objets afin de les regrouper selon un critère de similitude. La classification nécessite de pouvoir comparer les objets deux à deux. Il est donc possible de créer un graphe dont chaque objet est un sommet et chaque arête entre deux sommets évalue la similitude entre eux.

Les problèmes de classification peuvent donc être adaptés au partitionnement de graphe. De même, les méthodes de partitionnement de graphes peuvent être utilisées pour la classification [DHI07, DHI05, DIN01a, DHI01, ZHA01]. Il existe cependant beaucoup d'autres techniques pour résoudre les problèmes de classification [BER02].

Organisation du document

Ce document est découpé en deux grandes parties : la première décrit les méthodes de partitionnement de graphe que nous avons utilisées et développées pendant cette thèse ; la seconde présente les problèmes de partitionnement que nous avons cherché à résoudre, et les résultats que nous avons obtenus en les résolvant.

Cette thèse est composée de 7 chapitres qui se présentent comme suit :

Chapitre 1 : ce chapitre commence par quelques rappels de mathématiques et de théorie des graphes. La description formelle du partitionnement de graphe et de ses objectifs est suivie des présentations des problèmes de partitionnement contraint et non contraint, ainsi que de certaines notions générales relatives au partitionnement. Ce chapitre se termine sur la NP-difficulté des différents problèmes de partitionnement présentés ;

Partie I : méthodes de partitionnement de graphe ;

- **Chapitre 2 :** plusieurs méthodes classiques sont présentés dans ce chapitre. En premier, sont abordées les approches qui se sont montrées infructueuses pour le découpage aérien ; puis sont décrites les méthodes d'expansion de région, la méthode spectrale, l'algorithme de Kernighan-Lin et quelques unes de ces nombreuses améliorations, et la méthode multi-niveaux. Enfin, deux nouveaux algorithmes sont introduits, l'un pour le partitionnement, l'autre pour la répartition de charge ;
- **Chapitre 3 :** dans ce chapitre, trois métaheuristiques sont présentées, le recuit simulé, les algorithmes de colonies de fourmis et les algorithmes évolutionnaires. Pour chacune d'elle, l'adaptation que nous avons faite de celle-ci au partitionnement de graphe est décrite. Ce chapitre se termine par un état de l'art des adaptations les plus performantes de métaheuristiques pour le partitionnement de graphe contraint ;
- **Chapitre 4 :** la méthode de fusion-fission que nous avons développée y est présentée en quatre étapes : rappels de physique nucléaire ; bases de la méthode de fusion-fission ; adaptation au partitionnement non contraint ; adaptation au partitionnement contraint. Le chapitre se termine par une discussion sur la généralisation de cette méthode ;

Partie II : découpage de l'espace aérien et autres applications ;

Chapitre 5 : ce chapitre présente la problématique du découpage de l'espace aérien européen, modélise celle-ci sous la forme d'un problème de partitionnement non contraint, puis la résout au moyen de différentes méthodes présentées dans la première partie, et notamment de la fusion-fission ;

Chapitre 6 : la méthode de fusion-fission est utilisée pour résoudre les problèmes de partitionnement non contraint que sont la segmentation d'images et la classification de documents ;

Chapitre 7 : la méthode de fusion-fission appliquée au partitionnement contraint est comparée à de nombreuses autres méthodes sur des bancs de tests classiques du partitionnement contraint.

Le lecteur principalement intéressé par l'application de notre thèse au trafic aérien pourra commencer par lire les trois premières sections du chapitre 5 décrivant le problème du découpage de l'espace aérien, poursuivre par la lecture du chapitre 1 sur la présentation théorique du partitionnement de graphe, en omettant peut-être la dernière section sur la NP-difficulté du problème, puis continuer par la lecture des trois premières sections du chapitre 4 sur la méthode de fusion-fission, en faisant le lien lorsque le texte le proposera vers la méthode de percolation de la section 2.7.1. Il pourra alors retourner à la fin du chapitre 5, dans la section 5.4 présentant les résultats de l'application des méthodes de partitionnement au découpage de l'espace aérien. Enfin, s'il veut avoir un aperçu plus large des méthodes utilisées et adaptées, il pourra lire le reste de cette thèse dans son ordre natif.

Chapitre 1

Introduction au problème du partitionnement de graphe

De par l'étendue de ses applications, le partitionnement de graphe se décline en une multitudes de problèmes. Cependant, nous pouvons les regrouper en deux grandes catégories. La nature d'un problème du partitionnement de graphe est très différente selon que l'on cherche à obtenir des parties de tailles très proches ou sans contraintes de taille. Cette constatation est la pierre angulaire de la différentiation que nous faisons. Ainsi, nous étudierons deux problèmes : dans le cas où les parties sont de tailles semblables, on parlera de *partitionnement contraint*, dans le cas contraire, on parlera de *partitionnement non contraint*.

Avant de donner une définition précise de ces deux problèmes, ce chapitre présente quelques rappels de mathématiques (*infra* 1.1) et de théorie des graphes (*infra* 1.2) utilisés dans cette thèse. Ils seront suivi d'une description formelle du partitionnement de graphe (*infra* 1.3) et de ses objectifs (*infra* 1.4). Puis les deux problèmes de partitionnement seront présentés : le partitionnement contraint (*infra* 1.5) et le partitionnement non contraint (*infra* 1.6). Une avant-dernière section étudiera la NP-difficulté de ces problèmes de partitionnement (*infra* 1.8). La dernière section classera les différents problèmes de partitionnement que nous étudierons en fonction de leur caractère contraint ou non contraint (*infra* 1.9).

1.1 Notions mathématiques

Définition 1 (Cardinal d'un ensemble) *Soit X un ensemble fini d'éléments. Le cardinal de l'ensemble X est égal au nombre d'éléments de X, et on le note $card(X)$.*

En mathématiques, la partition d'un ensemble S d'objets se définit comme suit :

Définition 2 (Partition) *Soit un ensemble S quelconque. Un ensemble P de sous-ensembles de S est une partition de S si :*

1. *Aucun élément de P n'est vide ;*
2. *L'union des éléments de P est égal à S ;*
3. *Les éléments de P sont deux à deux disjoints.*

Les éléments de P sont appelés les parties de la partition P. Le cardinal de la partition P est alors le nombre de parties de P.

Comme nous l'avons vu, le problème du partitionnement de graphe fait partie des problèmes d'optimisation combinatoire, qui est une branche des mathématiques discrètes.

Les mathématiques discrètes sont l'étude des structures mathématiques où la notion de continuité est absente. Les objets étudiés en mathématiques discrètes sont des ensembles dénombrables. Dans le cadre des mathématiques discrètes, un ensemble dénombrable peut aussi être appelé ensemble discret.

Nous pouvons maintenant donner une définition formelle de ce qu'est un problème d'optimisation combinatoire :

Définition 3 (Problème d'optimisation combinatoire) *Un problème d'optimisation combinatoire se définit à partir d'un triplet* (E, p, f) *tel que :*
- E *est un ensemble discret appelé espace des solutions (aussi appelé espace de recherche) ;*
- p *est un prédicat sur* E*,* i.e. *une fonction de* E *dans* $\{vrai, faux\}$ *;*
- $f : E \longrightarrow R$ *associe à tout élément* $x \in E$ *un coût* $f(x)$*.* f *est appelée fonction objectif ou fonction de coût.*

p *permet de créer un ensemble* $E_a = \{x \in E \ tel \ que \ P(x) \ est \ vrai\}$*. L'ensemble* E_a *est appelé l'ensemble des solutions admissibles du problème.*
Il s'agit de trouver l'élément $\tilde{x} \in E_a$ *qui minimise* f *:*

$$f(\tilde{x}) = \min_{x \in E_a} f(x) \ .$$

Remarque : Le problème d'optimisation combinatoire consistant à chercher un élément maximum au lieu d'un élément minimum est de même nature puisque :

$$\max_{x \in E_a} f(x) = - \min_{x \in E_a} \left(-f(x)\right) \ .$$

Les problèmes d'optimisation combinatoire sont en général très coûteux à résoudre de façon optimale. C'est en particulier le cas du partitionnement de graphe, dont nous verrons, grâce à la théorie de la complexité, qu'il est NP-difficile (*infra* 1.8).

On peut décrire E comme l'ensemble des solutions du problème d'optimisation combinatoire que l'on cherche à résoudre quand on ne tient pas compte des contraintes de ce problème. Par exemple, le problème d'optimisation combinatoire, qui vise à trouver une partition des sommets d'un graphe $G = (S, A)$ en k parties de tailles égales (on choisit k diviseur de $card(S)$), aura pour ensemble de solutions E l'ensemble des partitions de S dont le nombre de parties va de un au nombre d'éléments de S, et dont les parties sont de tailles quelconques. Par contre, l'ensemble des solutions admissibles du problème, E_a, doit tenir compte des contraintes de celui-ci. Ainsi, dans notre exemple, l'ensemble E_a sera constitué des partitions de S en k parties de tailles égales.

Définition 4 (Optimum global, optimum local) *Soit un problème d'optimisation combinatoire* (E, p, f) *et* E_a *l'ensemble des solutions admissibles du problème induit par* p*. Soit* $\tilde{x} \in E_a$*.*
- *si l'on peut prouver que* $\forall x \in E_a, f(\tilde{x}) \leq f(x)$*, alors on dira que* $\tilde{x}$ *est l'optimum (minimum) global du problème ;*
- *s'il existe un ensemble* $V \subset E_a$*, contenant* $\tilde{x}$*, et au moins deux éléments, tel que* $\forall x \in V$*,* $f(\tilde{x}) \leq f(x)$*, alors on dira que* $\tilde{x}$ *est un optimum (minimum) local du problème.*

L'espace des solutions E dispose d'une « topologie ». Connaître les caractéristiques de celle-ci est très utile pour comprendre le but du fonctionnement des métaheuristiques (*infra* 3). Cette topologie résulte de la notion de proximité entre deux solutions, aussi appelées dans ce cas configurations. La distance entre deux configurations représente le nombre minimum de modifications élémentaires

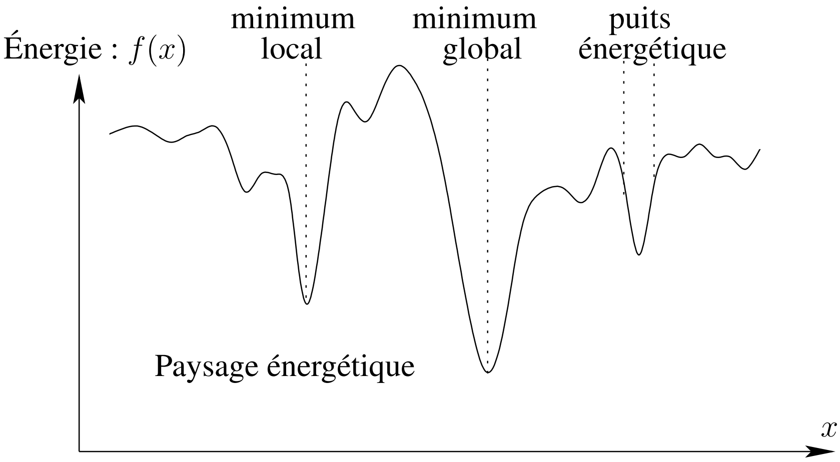


FIG. 1.1 – Paysage énergétique dans le cadre continu d'une fonction de coût pour un espace des solutions à une dimension.

pour passer de l'une à l'autre. De plus, puisqu'à chaque configuration x est associée une valeur $f(x)$, l'espace des solutions est caractérisé par une courbe à plusieurs dimensions appelée « paysage énergétique ». Dans ce paysage énergétique, les optima locaux ou globaux forment des « puits énergétiques » autour d'eux. Avant de décrire une solution du problème comme étant un minimum local, on vérifie en général que l'ensemble V est suffisamment « grand » par rapport à la taille de E_a. La figure 1.1 représente l'équivalent en continu du paysage énergétique d'une fonction de coût pour un espace des solutions à une dimension.

1.2 Graphes

Cette section rappelle quelques définitions de la théorie des graphes.

Définition 5 (Graphe) *Soient S un ensemble de $n_S \in \mathbb{N}^*$ éléments et A un ensemble de $n_A \in \mathbb{N}$ couples d'éléments de S. On appelle graphe G le couple (S, A). Les éléments de S sont appelés sommets du graphe et ceux de A, les arcs ou arêtes du graphe, suivant qu'ils sont orientés ou non.*

Lorsqu'il existe une relation entre au moins trois objets, on dit que ces objets sont reliés par une d'hyper-arête. Par extension, on ne parle plus de graphe, mais d'hypergraphe.

Définition 6 (Hypergraphe) *Soient un ensemble S d'éléments et un ensemble A de sous-ensembles non vides d'éléments de S. On appelle hypergraphe G le couple (S, A). Les éléments de A sont appelés hyper-arêtes.*

Lorsque les associations entre les objets ne sont pas réciproques, par exemple si l'objet a peut être en relation avec l'objet b, sans que l'objet b soit en relation avec a, on dit que les arêtes du graphe sont orientées.

Définition 7 (Graphe orienté, non orienté) *Soit un graphe $G = (S, A)$. Si $\forall (x, y) \in A$, $(y, x) \in A$, alors le graphe est dit non orienté et les éléments de A sont appelés arêtes du graphe. Dans ce cas, on note indifféremment une arête : $a \in A$, $(s, s') \in A$ avec s et s' dans S, ou encore (s', s). Dans le cas contraire, le graphe est dit orienté et les éléments de A sont appelés arcs du graphe.*

Dans cette thèse, nous n'étudierons que des graphes non orientés et pas d'hypergraphes.

Définition 8 (Degré d'un sommet) *Dans un graphe non orienté $G = (S, A)$, le degré d'un sommet $s \in S$ est le nombre d'arêtes auxquelles ce sommet appartient :*

$$deg(s) = card(\{(s, s') \in A, s' \in S\})$$

Lorsque les associations entre les objets sont quantifiées, on parle du poids des arêtes du graphe et on dit que le graphe est valué.

Définition 9 (Graphe valué ou pondéré) *Soit un graphe $G = (S, A)$. On dit que le graphe est valué (ou pondéré) si à chaque élément a de A est associé une valeur entière $poids(a) \in \mathbb{N}^*$.*

La valeur $poids(a)$ est appelée le poids de a.

Par extension, on considère qu'un couple de sommets $(s, s') \in S^2$ tel que $(s, s') \notin A$ possède un poids nul : $poids(s, s') = 0$.

Le poids d'un sous-ensemble X d'éléments de A, est la somme des poids des éléments de X : $poids(X) = \sum_{a \in X} poids(a)$.

De même, on associe parfois aux éléments s de S un entier strictement positif appelé le poids de s et noté $poids(s)$.

Remarque : Dans la théorie des graphes, le poids d'une arête ou d'un sommet peut être réel, mais comme nous ne nous intéressons dans cette thèse qu'aux problèmes discrets à valeurs positives, nous n'utiliserons que des poids entiers, strictement positifs. Le cas où une arête serait de poids nulle est considéré comme absurde car dans ce cas, l'arête est inexistante.

Il faut faire attention à ne pas confondre le cardinal d'un ensemble de sommets (respectivement d'arêtes) et le poids d'un ensemble de sommets (respectivement d'arêtes). Ceux-ci sont égaux si les sommets (respectivement les arêtes) ne sont pas pondérés.

Dans le cas des graphes valués, on redéfinit le degré d'un sommet $s \in S$ comme étant la somme des poids des arêtes adjacentes à ce sommet. Ainsi :

$$deg(s) = \sum_{(s,s') \in A} poids(s, s') .$$

À tout graphe non valué, on peut associer à chaque arête et à chaque sommet de ce graphe un poids unitaire.

Pour tout graphe $G = (S, A)$, on peut définir une bijection $u : S \rightarrow \{1, \ldots, n\}$ qui à tout élément de S associe un entier. La fonction u permet de numéroter les éléments de S. Pour plus de commodité par la suite, à tout élément de l'ensemble des sommets S d'un graphe G, on associera un rang dans S, et on confondra le sommet s_i de S avec son rang i dans S. Par exemple, pour tout $(s_i, s_j) \in A$, $poids(s_i, s_j)$ sera noté $poids(i, j)$.

Pour un graphe quelconque, on peut définir une unique fonction $poids : A \rightarrow \mathbb{N}$ qui à toute arête de ce graphe associe son poids. Cette fonction $poids$ peut être représentée sous la forme d'une matrice. On définit la matrice d'adjacence d'un graphe G comme étant la matrice associant à chaque arête de G son poids.

Définition 10 (Matrices d'adjacence et des degrés) *Soit un graphe $G = (S, A)$.*

La matrice M_{Adj} de dimension n_S^2, telle que $\forall (i, j) \in \{1, \ldots, n_S\}^2$:

$$(M_{Adj})_{ij} = \begin{cases} 0 & si\ i = j \\ p(i, j) & sinon \end{cases} ,$$

est appelée matrice d'adjacence du graphe G.

La matrice M_{Deg} de dimension n_S^2, telle que $\forall(i,j) \in \{1,\ldots,n_S\}^2$:

$$(M_{Deg})_{ij} = \begin{cases} deg(i) = \sum_{k=1}^{n_S} p(i,k) & si\ i = j \\ 0 & sinon \end{cases} ,$$

est appelée matrice des degrés du graphe G.

Définition 11 (Matrice Laplacienne) *Soit un graphe $G = (S, A)$. La matrice*

$$M_{Lap} = M_{Deg} - M_{Adj} ,$$

est appelée matrice Laplacienne (ou de Laplace) de G [MOH97].

Définition 12 (Adjacence) *Soient un graphe $G = (S, A)$ et une arête $a = (s, s') \in A$. On dit que les sommets s et s' sont les sommets adjacents de l'arête a. De même, on dit que a est l'arête adjacente des sommets s et s'.*

Définition 13 (Chemin dans un graphe) *Soient un graphe $G = (S, A)$, et deux sommets distincts $(s, s') \in S^2$. S'il existe une suite d'arêtes (ou d'arcs correctement orientés dans le cas d'un graphe orienté) permettant d'atteindre s' à partir de s, alors on dit qu'il existe un chemin de s vers s'.*

Définition 14 (Graphe connexe) *Soit un graphe $G = (S, A)$, on dit qu'il est connexe si, quels que soient les sommets s et s' de S, il existe un chemin de s vers s'.*

Un appariement d'un graphe est un ensemble d'arêtes de ce graphe qui n'ont pas de sommets en commun.

Définition 15 (Appariement) *Soit un graphe $G = (S, A)$, l'appariement M du graphe G est un ensemble d'arêtes non-adjacentes deux à deux.*

On dit que l'appariement est maximum lorsqu'il contient le plus grand nombre possible d'arêtes.

On dit que l'appariement M est maximal lorsque toute arête du graphe possède une intersection non vide avec au moins une arête de M.

Remarque : Tout appariement maximal est aussi un appariement maximum. Soit un appariement maximal M, si une arête quelconque de A qui n'est pas dans M est ajoutée à M , alors M n'est plus un appariement de G. La réciproque n'est pas nécessairement exacte : soit un graphe linéaire de 4 sommets, l'appariement composé de l'arête formé des deux sommets centraux est maximal mais pas maximum. Un graphe peut posséder plusieurs appariements maximal (et *a fortiori* plusieurs appariements maximum).

1.3 Description formelle du problème du partitionnement de graphe

Avant de présenter ce qu'est le problème général du partitionnement de graphe, il nous faut définir ce qu'est la partition d'un graphe. Comme nous l'avons vu (*supra* 1.2), un graphe est un couple formé d'un ensemble de sommets et d'un ensemble d'arêtes. Il est donc possible de faire la partition, au sens mathématique (*supra* 1.1), de l'ensemble des sommets comme de l'ensemble des arêtes. Cependant, bien que certains problèmes cherchent à partitionner les arêtes d'un graphe [HOL81], on entend le plus souvent par partition d'un graphe, la partition des sommets de ce graphe.

Définition 16 (Partition des sommets d'un graphe) *Soient un graphe $G = (S, A)$ et un ensemble de k sous-ensembles de S, noté $P_k = \{S_1, \ldots, S_k\}$. On dit que P_k est une partition de G si :*
- *aucun sous-ensemble de S qui est élément de P_k n'est vide :*
$$\forall i \in \{1, \ldots, k\}, S_i \neq \emptyset \ ;$$
- *les sous-ensembles de S qui sont éléments de P_k sont disjoints deux à deux :*
$$\forall (i, j) \in \{1, \ldots, k\}^2, i \neq j, S_i \cap S_j = \emptyset \ ;$$
- *l'union de tous les éléments de P_k est S :*
$$\bigcup_{i=1}^{k} S_i = S \ .$$

Les éléments S_i de P_k sont appelés les parties de la partition.
Le nombre k est appelé le cardinal de la partition, ou encore le nombre de parties de la partition.

La définition de la partition des sommets d'un graphe est peu restrictive sur la taille des parties. Pour remédier à cette lacune et ainsi pouvoir mesurer la taille des parties d'une partition, la notion de « balance » de partitionnement est introduite. Le terme balance est un néologisme de l'anglais *balance*, qui signifie en français *équilibre*. Le mot balance a été retenu dans cette thèse pour les raisons suivantes : il signifie la « mesure du poids », alors qu'équilibre désigne une « position stable ». Or si la « stabilité » d'une partie est une notion obscure, le poids d'une partie a un sens concert (dans le cas d'un graphe non pondéré c'est son cardinal). Ainsi, comme le terme partitionnement fait référence à la création de parties, nous préférons parler de *balance de partitionnement*, et d'*équilibre de la partition*.

Définition 17 (Balance de partitionnement) *Soient un graphe $G = (S, A)$ et une partition $P_k = \{S_1, \ldots, S_k\}$ de ce graphe en k parties. Le poids moyen d'une partie S_i de P_k est*
$$poids_{moy} = \left\lceil \frac{poids(S)}{k} \right\rceil \ ,$$
où $\lceil x \rceil \in R$ désigne le premier entier supérieur ou égal à x.
La balance $bal(P_k)$ de la partition P_k est égale à la division du poids de la partie de poids maximal de P_k par le poids moyen d'une partie :
$$bal(P_k) = \frac{\max_i poids(S_i)}{poids_{moy}} \ .$$

Ainsi, une partition dont la balance de partitionnement est unitaire a des parties de poids identiques, à une unité près.

Comme nous l'avons vu, le problème du partitionnement de graphe fait partie des problèmes d'optimisation combinatoire ; il est donc naturel de l'introduire sous cette forme. Présenté simplement, le problème général du partitionnement de graphe consiste à trouver une partition d'un graphe qui satisfasse un prédicat p et qui minimise une fonction objectif f. Voici une définition plus formelle :

Définition 18 (Problème général du k-partitionnement de graphe) *Soit un graphe $G = (S, A)$. Nous allons définir le triplet (E, p, f) caractérisant le problème général du k-partitionnement de graphe :*

- *l'espace des solutions E est défini comme l'ensemble des partitions de S (ces partitions ont un cardinal compris entre 1 et $card(S)$;*
- *soit un entier $k \geq 2$ et p' un prédicat défini par $P \in E$, $p'(P) = vrai \iff card(P) = k$. Le prédicat p est défini sur E tel que $p(P) = vrai \Rightarrow p'(P) = vrai$;*
- *soit $f : E \longrightarrow R$ une fonction objectif.*

L'ensemble des solutions admissibles du problème est défini par :

$$E_a = \{P \in E \text{ tel que } p(P) = vrai\} \ .$$

Le problème général du partitionnement de graphe consiste à trouver la partition $\tilde{P_k} \in E_a$ qui minimise f :

$$f(\tilde{P_k}) = \min_{P_k \in E_a} f(P_k) \ .$$

Par définition d'un graphe, l'ensemble S est fini. Le nombre de partitions d'un ensemble à n éléments est appelé le nombre de Bell relatif à n, et noté B_n. C'est le nombre de relations d'équivalences distinctes sur un ensemble à n éléments. On pose $B_0 = 1$. Le nombre de Bell vérifie la relation de récurrence suivante :

$$B_{n+1} = \sum_{k=0}^{n} C_n^k B_k \ ,$$

où $C_n^k = \frac{n!}{k!(n-k)!}$ est le coefficient binomial de n et k, c-à-d le nombre de parties à k éléments d'un ensemble de n éléments.

Le nombre de partitions en k parties d'un ensemble à n éléments est quant à lui appelé le nombre de Stirling de seconde espèce. Ce nombre est noté $S_{n,k}$ et vaut :

$$S_{n,k} = \sum_{i=0}^{k} (-1)^i \frac{(k-i)^n}{i!\,(k-i)!} \ .$$

Il existe une relation entre le nombre de Bell et le nombre de Stirling de seconde espèce :

$$B_n = \sum_{i=1}^{n} S_{n,i} \ .$$

Ainsi, l'ensemble E est un ensemble fini. C'est donc bien un ensemble discret comme le veut la définition d'un problème d'optimisation combinatoire (*supra* 1.1).

Le problème général du partitionnement de graphe se définit en fonction d'un couple prédicat, fonction objectif : (p, f). Ainsi, le partitionnement de graphe peut se diviser en de nombreux sous-problèmes, suivant la nature de p et de f. Cependant, ces problèmes peuvent se classer en 2 catégories distinctes selon la nature du prédicat p (*infra* 1.5 et 1.6).

1.4 Fonctions objectifs pour le partitionnement de graphe

Les différentes fonctions objectifs pour le partitionnement de graphe s'articulent toutes autour de deux concepts : le coût de coupe entre les parties de la partition et le poids de ces parties.

Soit un graphe $G = (S, A)$. Soient deux sous-ensembles $S_a \subseteq S$ et $S_b \subseteq S$, on définit le coût de coupe entre ces deux sous-ensembles par :

$$coupe(S_a, S_b) = \sum_{u \in S_a, v \in S_b} poids(u, v) \ .$$

Dans le reste de cette section, on choisit une partition $P_k = \{S_1, \ldots, S_k\}$ de S en k parties. Les fonctions objectifs présentées seront définies par rapport à cette partition.

La plus simple des fonctions objectifs utilisée en partitionnement de graphe est appelée le coût de coupe d'une partition, ou en anglais, *cut*. Elle cherche à minimiser la somme des poids des arêtes entre les parties de la partition P_k. Elle est déjà utilisée par Brian Kernighan et Shen Lin dans [KER70] :

$$\begin{aligned} coupe(P_k) &= \sum_{i<j} coupe(S_i, S_j) \\ &= \frac{1}{2} \sum_{i=1}^{k} coupe(S_i, S - S_i) \ . \end{aligned}$$

La fonction objectif qui vise à minimiser pour chaque partie le rapport entre son coût de coupe et son poids est appelée ratio de coupe, en anglais *ratio cut*. Elle est introduite par Yen-Chuen Wei et Chung-Kuan Cheng dans [WEI89] :

$$ratio(P_k) = \sum_{i=1}^{k} \frac{coupe(S_i, S - S_i)}{poids(S_i)} \ .$$

Cependant, quand le terme ratio de coupe est utilisé, il faut être averti que d'autres fonctions objectifs sont aussi appelées du même nom dans la littérature [WAN03].

La troisième et dernière fonction largement utilisée pour le partitionnement de graphe est appelée coût normalisé, ou *normalized cut* en anglais. Cette fonction, qui est la plus récente, a été présentée par Jianbo Chi et Jitendra Malik dans [SHI00]. Elle cherche à minimiser, pour chaque partie, le rapport entre son coût de coupe et la somme du poids des arêtes adjacentes à au moins un de ses sommets. Autrement dit, elle cherche à minimiser, pour chaque partie, le rapport entre la somme du poids des arêtes adjacentes à exactement un de ses sommets et la somme du poids des arêtes adjacentes à au moins un de ses sommets :

$$\begin{aligned} norm(P_k) &= \sum_{i=1}^{k} \frac{coupe(S_i, S - S_i)}{coupe(S_i, S)} \\ &= \sum_{i=1}^{k} 1 - \frac{coupe(S_i, S_i)}{coupe(S_i, S)} \ . \end{aligned}$$

La fonction de coût normalisé permet d'isoler les régions du graphe dont les sommets sont très liés entre eux. Ses auteurs l'ont créée pour remplacer le ratio de coupe dans leur problème de segmentation d'image.

Remarques :

- dans le cas des graphes étudiés dans cette thèse qui sont pondérés par des poids strictement positifs, toutes les fonctions objectifs présentées ci-dessus sont positives ;
- un certain nombre de graphes utilisés dans les problèmes de partitionnement ont comme poids de chacun de leurs sommets, la somme des poids des arêtes adjacentes à ce sommet. Ce qui veut dire que $\forall i \in \{1, \ldots, k\}, coupe(S_i, S) = poids(S_i)$, et donc que le ratio de coupe est égal au coût normalisé pour une partition d'un tel graphe, *i.e.* $\forall P_k \in E, ratio(P_k) = norm(P_k)$.

Toutes ces fonctions objectifs permettent de définir l'« énergie » d'une partie d'une partition. Le terme d'énergie est utilisé en référence à la notion de paysage énergétique présenté en section 1.1.

Cette énergie correspond à la valeur apportée par la partition à la fonction objectif. Soit une partition $P_k = \{S_1, \ldots, S_k\}$ de S en k éléments. Dans le cas du coût de coupe, l'énergie d'une partie S_i sera :

$$\acute{e}nergie_{cc}(S_i) = coupe(S_i, S - S_i) \ .$$

Dans le cas du ratio de coupe, l'énergie d'un partie S_i sera :

$$\acute{e}nergie_{rc}(S_i) = \frac{coupe(S_i, S - S_i)}{poids(S_i)} \ .$$

Dans le cas de la coupe normalisée, l'énergie d'un partie S_i sera :

$$\acute{e}nergie_{cn}(S_i) = \frac{coupe(S_i, S - S_i)}{coupe(S_i, S)} \ .$$

1.5 Le partitionnement contraint

La littérature classique du partitionnement de graphe s'attache au problème du partitionnement contraint. C'est ce problème que résolvent la plupart des outils de partitionnement de graphe tels que Metis, Jostle, Chaco, Scotch ou Party, comme nous le verrons plus loin (*infra* A).

L'un des premiers papiers à présenter ce problème fût celui de Brian Kernighan et Shen Lin [KER70]. C'est aussi ce problème qui sert le plus souvent à illustrer les nouvelles méthodes de partitionnement de graphe. C'est le cas pour la méthode spectrale présentée par Alex Pothen, Horst Simon et Kang-Pu Liou [POT90], ainsi que pour la méthode multi-niveaux introduite par Stephen Barnard et Horst Simon [BAR93, BAR94].

Le partitionnement contraint sert, entre autres, comme nous l'avons vu précédemment, à résoudre des problèmes d'ingénierie, de calcul haute performance, de résolution de systèmes linéaires, de maillage, et dans certains cas, de conception de circuits intégrés. Cependant, certaines critiques ont été formulées concernant l'utilisation de cette approche pour modéliser ces problèmes, c'est notamment le cas dans [HEN98] .

Dans la littérature, le partitionnement de graphe contraint est appelé *multi-way graph partitioning* [YAR00] ou *k-way graph partitioning* [KAR98d].

Le problème du partitionnement de graphe contraint consiste à trouver une partition en k parties qui minimise une fonction objectif f et dont la balance de partitionnement soit unitaire, *i.e.* les parties doivent avoir le même poids, à une unité près.

Or, si une partition P_k a des parties de même poids, alors son ratio de coupe (*infra* 1.4) devient égal à son coût de coupe, à une constante multiplicative près. En effet, dans ce cas :

$$\begin{aligned} ratio(P_k) =& \sum_{i=1}^{k} \frac{coupe(S_i, S - S_i)}{poids(S_i)} \\ =& \frac{k}{poids(S)} \sum_{i=1}^{k} coupe(S_i, S - S_i) \\ =& \frac{k}{poids(S)} coupe(P_k) \ . \end{aligned}$$

Ainsi, pour le problème du partitionnement de graphe contraint, minimiser le ratio de coupe est équivalent à minimiser le coût de coupe (remarque : si k n'est pas diviseur de $poids(S)$, alors les deux minimisations de ces fonctions restent cependant très proches).

De plus, dans le cas où est cherchée une partition P_k dont les parties ont le même poids, l'utilisation du coût normalisé perd son sens. En effet, cette fonction de coût a été créée pour isoler des autres régions celles dont les sommets sont très liés, ce qui va le plus souvent dans un sens contraire à la recherche de parties de tailles égales. Cette fonction de coût n'est donc jamais utilisée dans les problèmes de partitionnement contraint.

Ainsi, le seul réel objectif du partitionnement de graphe contraint est de minimiser le coût de coupe des arêtes entre les parties du graphe. Ceci nous permet de donner sa définition :

Définition 19 (Partitionnement contraint) *Soient un graphe $G = (S, A)$, un nombre de parties k et une balance de partitionnement maximale, $balmax$. Soit l'espace des solutions E, défini par le problème général du partitionnement de graphe. L'ensemble E_a des solutions admissibles du problème est défini par :*

$$E_a = \{P_i \in E \text{ tel que } card(P_i) = k \text{ et } bal(P_i) \leq balmax\} .$$

Le problème du partitionnement contraint consiste à trouver $\tilde{x} \in E_a$ qui minimise la fonction de coût de coupe :

$$coupe(\tilde{x}) = \min_{x \in E_a} coupe(x) .$$

Remarques :

- la balance de partitionnement maximale des problèmes de partitionnement contraint est souvent très petite : $balmax \in [1,0; 1,05]$;
- dans les problèmes de partitionnement contraint, une petite relaxation de la balance de partitionnement est souvent acceptée, car il est connu qu'avec une valeur de la balance légèrement supérieure à l'unité, peuvent être trouvées des partitions de coût de coupe bien inférieures, sans que cela soit techniquement nuisible à la résolution du problème [SIM97].

1.6 Le partitionnement non contraint

Le problème du partitionnement non contraint est un problème un peu plus « exotique » que celui du partitionnement contraint. Nous utilisons le terme exotique car ce problème se dérive en de nombreux problèmes très proches les uns des autres et dont la fonction objectif est régulièrement améliorée, ou tout du moins changée. Si le premier but du partitionnement contraint consiste à trouver une partition respectant une balance très restrictive, celui du partitionnement non contraint est de minimiser une fonction objectif. Alors que le second but du partitionnement contraint et la minimisation de la fonction de coût, le partitionnement contraint n'en a en général pas.

Le partitionnement non contraint est assez proche dans sa formulation du partitionnement de données ou *clustering* en anglais. Le problème du partitionnement de données est de regrouper des éléments similaires dans des ensembles les plus distincts possibles. Pour comparer deux éléments, la notion de distance entre eux est utilisée. Ce qui ramène le problème du partitionnement de données à créer des sous-ensembles d'éléments les plus distants possibles les uns des autres. Or, on peut facilement fabriquer un graphe dont les sommets sont ces éléments et dont les arêtes ont pour poids l'inverse de la distance entre deux éléments. Dans le cas où le nombre de parties cherchées est fixe, un problème de partitionnement de données se ramène à un problème de partitionnement non contraint. Il en résulte que de nombreux travaux ont cherché à résoudre des problèmes de partitionnement de données à l'aide d'outils de partitionnement de graphe.

Le partitionnement non contraint est très utilisé pour la segmentation d'images [SHI98b, SHI98a, PER98, MEI00, SHI00, GDA01, WAN01, MAR04, BEN05], mais aussi pour la classification de textes [DHI01, ZHA01, DIN01a, ZHA04]. Dans le cas de la segmentation d'images, de nombreux articles comparent différentes fonctions objectifs et proposent de nouvelles fonctions répondant à de nouveaux critères [SHI98b, SHI00, PER98, GDA01, WAN01, FEL04]. D'autres problèmes plus originaux, comme le découpage de l'espace aérien (*supra* 5), sont aussi des problèmes de partitionnement non contraint. Enfin, un certain nombre d'articles résolvent des problèmes de partitionnement de circuit VLSI par une approche de partitionnement de graphe non contraint [HAG92b, RIE94, ALP95a]. Cependant, dans le cas des circuits VLSI, ceux-ci sont dans la majorité des cas considérés comme des hypergraphes plutôt que des graphes [KER70].

Les fonctions objectifs que cherche à minimiser un problème de partitionnement non contraint sont variées. La section 1.4 présente le ratio de coupe et le coût normalisé. Ces deux fonctions objectifs, avec quelques variantes très proches, sont celles qui reviennent le plus souvent dans les problèmes de partitionnement non contraint. Dans certains cas, et dans une certaine mesure, on peut considérer pour le partitionnement non contraint que la contrainte sur la balance de partitionnement est intégrée à la fonction objectif. Ainsi, contrairement au cas contraint, la plupart des problèmes omettent la contrainte sur la balance de partitionnement et n'énoncent que la fonction objectif à minimiser. La contrainte sur la balance de la partition ne sera donc pas incluse dans la définition du partitionnement non contraint, mais pourra être ajoutée par la suite :

Définition 20 (Partitionnement non contraint) *Soient un graphe $G = (S, A)$ et un nombre de parties k. Soit l'espace des solutions E, défini par le problème général du partitionnement de graphe. L'ensemble E_a des solutions admissibles du problème est défini par :*

$$E_a = \{P_i \in E \text{ tel que } card(P_i) = k\} .$$

Le problème du partitionnement contraint consiste à trouver $\tilde{x} \in A$ qui minimise une fonction de coût f comme le coût normalisé ou le ratio de coupe :

$$f(\tilde{x}) = \min_{x \in A} f(x) .$$

Comme l'ensemble des solutions admissibles du problème dans le cas du partitionnement contraint est bien plus petit que dans le cas non contraint, trouver une partition qui satisfasse ce dernier est plus facile que pour le premier. Dans ce dernier cas, un algorithme qui s'écarte de la balance requise aura beaucoup de mal à retrouver un partition la vérifiant. Il est donc difficile de commencer une recherche hors de l'espace des solutions admissibles dans le but d'y revenir par la suite. Ainsi, il peut sembler plus simple de trouver une bonne solution avec une balance faible qu'une balance forte. Cependant il n'en est rien, car s'il est plus facile de trouver une solution admissible dans le cas non contraint, c'est que l'espace admissible est plus grand, mais il n'est pas plus facile à explorer. De plus, un minimum local dans le cas contraint n'est pas forcément proche d'un minimum local du cas non contraint.

Dans une certaine mesure et dans le cas où une balance maximale est imposée, le partitionnement non contraint peut être vu comme un problème d'optimisation multiobjectif consistant à minimiser une des fonctions objectifs de la section 1.4 et à minimiser la balance de partitionnement [RUM02, GIL06, SEL03, SEL06]. Pour pallier les lacunes des fonctions objectifs classiques (*supra* 1.4) et intégrer d'autres objectifs pouvant être partiellement antagonistes, d'autres articles traitent du problème de partitionnement de graphe comme un problème multiobjectif [SCH99].

Remarque : On peut transformer le problème du partitionnement de graphe non connexe en partitionnements de graphes connexes par le mécanisme suivant : dans le cas d'un graphe non connexe, une partition évidente de ce graphe est l'ensemble formé des différentes parties connexes du graphe. De plus, une telle partition a un coût de coupe nul, ce qui entraîne que les différentes fonctions objectifs classiques présentées plus haut (*supra* 1.4) prennent toutes une valeur nulle pour un tel graphe.

1.7 Généralités liées au partitionnement

1.7.1 Différences entre partitionnement contraint et non contraint

Nous avons présenté dans les sections précédentes deux problèmes : le partitionnement contraint (*supra* 1.5) et le partitionnement non contraint (*supra* 1.6). Cependant, bien que ces deux problèmes aient une formulation assez proche, les algorithmes utilisés pour les résoudre sont en général bien différents, si ce n'est dans leurs principes, du moins dans leurs implémentations et leurs buts. En effet, au risque de trop schématiser, ces deux problèmes diffèrent principalement dans leurs priorités :

– **le partitionnement contraint** a pour but premier de trouver des parties de poids identiques et comme second but de diminuer le coût de coupe ;
– **le partitionnement non contraint** a pour but de minimiser le coût de coupe dans la limite de parties dont la taille est inversement proportionnelle à ce coût de coupe. Le principe étant d'éviter un déséquilibre trop grand des parties, sauf dans le cas où un coût de coupe est vraiment extrêmement faible. Ainsi, le but premier du cas non contraint est de minimiser le coût de coupe, et son second but est d'équilibrer les parties.

L'inversion de l'ordre des priorités entre ces deux problèmes entraîne qu'un algorithme permettant de traiter un des deux problèmes va donner des résultats très médiocres dans le cas du second problème. C'est ce que l'on a pu constater dans [BIC06b, BIC07].

Il faut noter l'absence d'article dans la littérature abordant ces deux problèmes de front et les comparant exhaustivement. Cependant, les problèmes de partitionnement non contraint sont multiples et des efforts ont été réalisés par Inderjit S. Dhillon pour les synthétiser [DHI04b, DHI07].

1.7.2 Affinage et répartition de charge d'une partition

Le problème du partitionnement est un problème d'optimisation combinatoire. Pour trouver une solution minimisant la fonction objectif, un algorithme d'optimisation globale est souvent utilisé. Un tel algorithme permet de trouver une solution dans un puit énergétique « profond ». Cependant, celle-ci n'est pas toujours la solution d'énergie minimale de ce puits. Dans ce ca, un algorithme d'optimisation locale va permettre de trouver une solution de coût inférieur. Un tel algorithme est appelé algorithme d'*affinage* (le terme raffinage est aussi utilisé), et permet d'affiner une partition. De même, la partition trouvée par l'algorithme d'optimisation globale peut ne pas respecter la balance de partitionnement. Dans ce cas, un algorithme de répartition de charge sera utilisé pour répartir les sommets de façon à ce que la balance de partitionnement soit satisfaite.

Les algorithmes d'affinage et de répartition de charge ont en commun deux objectifs :

1. trouver une partition telle que la fonction objectif prend la valeur la plus faible possible ;
2. trouver une partition telle que la balance de partitionnement soit autant respectée que possible.

La différence entre ces deux méthodes vient de l'ordre des priorités entre ces deux objectifs. L'affinage tient pour plus important de minimiser la fonction objectif, quand la répartition de charge aura comme but premier de respecter la balance de partitionnement.

Ces deux objectifs sont en général antinomiques [SIM97]. Cependant leur conciliation est primordiale car à la base même du partitionnement. Ainsi, suivant que la partition obtenue respecte ou non la balance de partitionnement, l'un ou l'autre des algorithmes est utilisé. Cependant, après avoir appliqué un algorithme de répartition de charge à une partition, celle-ci est en générale affinée. Différents algorithmes d'affinage basés sur celui de Kernighan-Lin sont présentés en sections 2.4 et 2.5. Un algorithme de répartition de charge est introduit en sous-section 2.5.4 et un autre est décrit en sous-section 2.7.2.

1.7.3 Technique de la bissection récursive

Algorithme 1 Algorithme récursif de bissection de graphe.

```
Procédure BISSECTIONREC(G = (S, A), k)
    P_k = {S_1, ..., S_k}  % Partition de G en k parties
    Procédure ITÉRER(S', k', num)
        si k' > 1 alors
            k'_1 ← ⌊k'/2⌋  % entier le plus grand inférieur à k'/2
            k'_2 ← k' − k'_1
            S'_1, S'_2 ← bissection(S', k'_1/k', k'_2/k')
            ITÉRER(S'_1, k'_1, num)
            ITÉRER(S'_2, k'_2, num + k'_1)
        sinon  % Plus de bissection à effectuer, la partie S' est mise dans P_k
            S_num ← S'
        fin si
    fin Procédure
    Itérer(S, k, 1)
    retourner P_k
fin Procédure
```

Beaucoup d'algorithmes de partitionnement ne s'intéressent qu'au cas de la bissection de graphes [RON05, MAR05, MAR06, CHA07]. Lorsqu'il est possible d'ajuster précisément la balance de partitionnement, la technique de la bissection récursive permet d'adapter ces algorithmes au k-partitionnement. Cependant, lorsqu'il n'est pas possible de régler la balance de partitionnement, la technique de la bissection récursive ne permet de faire que des 2^i-partitionnements ($i \in \mathbb{N}$).

L'algorithme 1 présente la méthode d'adaptation d'un algorithme, que l'on nommera $bissection$, au k-partitionnement. L'algorithme 1 utilise une fonction récursive, ITÉRER, prenant en paramètres la partie S' du graphe à partitionner, le cardinal k' de la partition de S', ainsi que le rang num de la première partie de la partition de S' dans la partition finale P_k. La valeur prise par k'_1 est l'entier le plus grand inférieur à $\frac{k'}{2}$, celle prise par k'_2 correspond au reste du nombre de parties à trouver. La fonction $bissection$ doit trouver une bissection de S' en deux parties S'_1 et S'_2 telles que :

$$poids(S'_1) \leq \frac{k'_1}{k'} poids(S') \quad \text{et} \quad poids(S'_2) \leq \frac{k'_2}{k'} poids(S') \ .$$

Pour que la partition finale respecte la balance de partitionnement, plusieurs solutions existent. La plus simple consiste à chercher à chaque itération une bissection du graphe en 2 parties de tailles égales. La partition trouvée sera alors parfaitement balancée, *i.e.* $bal(P_k) = 1,00$. Lorsqu'un algorithme d'affinage sera appliqué à la partition P_k, sa balance pourra alors évoluer. Une seconde solution consiste à être très permissif sur la balance de partitionnement dès le départ, puis à la durcir en fonction de la taille des parties déjà obtenues. Cette solution est plus compliquée car il faut après chaque bissection recalculer la balance maximale que devront respecter les bissections de chacune des deux parties trouvées, afin qu'au final la partition respecte bien la balance demandée.

Malgré l'intérêt de pouvoir adapter les méthodes de bissection au k-partitionnement, il a été montré dans [SIM97] que l'algorithme récursif de bissection ne pouvait explorer certaines solutions du problème. Celles-ci ne peuvent être trouvées qu'au moyen de méthodes de k-partitionnement direct, d'où l'intérêt de ces dernières.

1.8 NP-difficulté des problèmes de partitionnement

1.8.1 Le cas du partitionnement contraint

Pour montrer qu'un problème d'optimisation est NP-difficile, il suffit de montrer que le problème de décision associé est NP-complet. Or, le problème de décision associé au problème du partitionnement de graphe contraint (*supra* 1.5) peut se formaliser de la façon suivante :

Problème de décision associé au problème du partitionnement de graphe contraint :

Instance du problème. Soit un graphe $G = (S, A)$ pondéré sur ses sommets par $\forall s \in S$, $poids(s) \in \mathbb{N}^*$ et sur les arêtes par $\forall a \in A$, $poids(a) \in \mathbb{N}^*$. Soient $poids_{max} \in \mathbb{N}^*$ et $coupe_{max} \in \mathbb{N}^*$.

Question. Existe-t-il une partition P_k de S en k parties disjointes $S_1, \ldots, S_k$ telle que pour toute partie S_i,

$$\sum_{s \in S_i} poids(s) \leq poids_{max}$$

et telle que :

$$coupe(P_k) \leq coupe_{max} \text{ ?}$$

Solution. Ce problème reste NP-complet pour $poids_{max} \geq 3$, même lorsque le poids des sommets et des arêtes est unitaire. Il peut être résolu en temps polynomial quand $poids_{max} = 2$.

Ce problème de décision, appelé *graph partitioning*, est traité dans [HYA73] et d'une manière qui fait référence dans [GAR79]. Ainsi, le problème du partitionnement de graphe contraint est NP-difficile.

1.8.2 Le cas du partitionnement non contraint

Le problème du partitionnement de graphe non contraint est défini en section 1.6. Le but de cette sous-section est de montrer que ce problème est NP-difficile pour les fonctions objectifs de coupe normalisée et de ratio de coupe.

Pour montrer qu'un problème d'optimisation est NP-difficile, nous allons montrer que le problème de décision associé, noté Π, est NP-complet. Pour montrer que Π est NP-complet, d'après [GAR79], il suffit de suivre les étapes suivantes :

1. montrer que Π est dans NP ;
2. trouver un problème Π' NP-complet proche de Π ;
3. réduire Π' à Π : trouver un sous-problème de Π' équivalent à Π.

NP-difficulté de la coupe normalisée

La NP-difficulté du problème d'optimisation du partitionnement de graphe non contraint pour la coupe normalisée a été montrée dans [SHI00] par réduction au problème de partition. Le problème de partition, NP-complet, s'énonce ainsi :

Problème de décision associé au problème de partition :

Instance du problème. Soient un ensemble S fini et une fonction de poids pour tous les éléments $s \in S$: $poids(s) \in \mathbb{N}^*$.

Question. Existe-t-il un sous-ensemble $S' \subseteq S$ tel que $\sum_{s \in S'} poids(s) = \sum_{s \in S - S'} poids(s)$?

La démonstration de la NP-difficulté du problème de partitionnement utilisant la coupe normalisée repose sur la construction d'un graphe bien particulier. La propriété de ce graphe est qu'une bissection de ce graphe a une coupe normalisée suffisamment petite si et seulement si on peut trouver un sous-ensemble de S dont la somme des poids est égale à la moitié du poids de S. Le problème d'existence de la coupe normalisée se ramène alors à celui de l'existence de la partition, ce qui achève la démonstration.

NP-difficulté du ratio de coupe

Le problème Π d'existence associé au ratio de coupe peut se formaliser de la façon suivante :

Problème de décision associé au problème du ratio de coupe :

Instance du problème. Soit un graphe $G = (S, A)$ pondéré sur les sommets par $\forall s \in S,\ poids(s) \in \mathbb{N}^*$ et sur les arêtes par $\forall a \in A,\ poids(a) \in \mathbb{N}^*$. Soit $coupe_{max} \in \mathbb{N}^*$.

Question. Existe-t-il une partition P_k de S en k sous-ensembles $S_1, \ldots, S_k$ disjoints, non vides, telle que $ratio(P_k) \leq coupe_{max}$?

Pour montrer que Π est dans NP, il suffit de montrer que la vérification d'une solution au problème Π s'effectue en un temps polynomial. Soit une solution P_k au problème Π. Vérifier que les ensembles $S_i \in P_k$ sont disjoints et inférieurs à un poids maximal se fait en $O(n_S)$. Le calcul du ratio de coupe se fait en parcourant les arêtes, *i.e.* en $O(n_A)$. Vérifier que chaque ensemble S_i est non vide est une opération en $O(k)$. Ainsi, vérifier qu'une partition est solution de Π se fait en temps polynomial.

Il existe plusieurs méthodes permettant de prouver la NP-complétude d'un problème d'optimisation [GAR79]. La méthode la plus commune consiste à restreindre le problème de départ pour montrer qu'un cas particulier de celui-ci est un problème NP-complet.

Le problème de la coupe minimale dans des ensembles finis, intitulé *minimum cut into bounded sets* dans [GAR79], est proche du problème Π. Il s'énonce ainsi :

Problème de décision associé au problème de la coupe minimale :

Instance du problème. Soit un graphe $G = (S, A)$ pondéré sur les sommets par $\forall s \in S$, $poids(s) \in \mathbb{Z}^+$ et sur les arêtes par $\forall a \in A$, $poids(a) \in \mathbb{Z}^+$. Soient s_1 et s_2 deux sommets différents de S. Soient $poids_{max} \leq poids(S)$ et $coupe_{max}$ deux entiers positifs.

Question. Existe-t-il une bissection P_2 de S en deux sous-ensembles S_1, S_2 disjoints, vérifiant $s_1 \in S_1$, $s_2 \in S_2$, $poids(S_1) \leq poids_{max}$ et $poids(S_2) \leq poids_{max}$, telle que $coupe(S_1, S_2) \leq coupe_{max}$?

Montrons maintenant que Π est NP-complet :

Les ensembles S_1 et S_2 sont non vides et disjoints. De plus, leur poids est borné. Supposons que

$$poids(S_1) \geq poids(S_2) \ .$$

Pour $k = 2$, le ratio de coupe devient :

$$ratio(S_1, S_2) = \frac{coupe(S_1, S_2)}{poids(S_1)} + \frac{coupe(S_2, S_1)}{poids(S_2)} \ .$$

Puisque S_1 et S_2 son non vides, $poids(S_2) \geq 1$, puis

$$0 < \frac{1}{poids(S_1)} \leq \frac{1}{poids(S_2)} \leq 1 \ ,$$

et donc

$$\begin{aligned} ratio(S_1, S_2) &\leq 2\frac{coupe(S_1, S_2)}{poids(S_2)} \\ &\leq 2coupe(S_1, S_2) \\ &\leq 2coupe_{max} \ . \end{aligned}$$

Ainsi, Π est NP-complet.

Le problème d'optimisation du partitionnement de graphe non contraint pour le ratio de coupe est donc NP-difficile.

1.9 Conclusion

Ce chapitre a commencé par rappeler quelques notions d'optimisation et de théorie des graphes qui serviront dans cette thèse. Puis, le problème général du partitionnement de graphe et plusieurs fonctions objectifs utilisables par ce problème ont été décrits. Comme ce problème se divise en deux sous-problèmes, le partitionnement « contraint » et le partitionnement « non contraint », ceux-ci sont présentés de manière formelle. Les notions d'affinage et de répartition de charge, liées au problème général du partitionnement, ont été introduites, ainsi que la technique de la bissection récursive. Enfin, la NP-difficulté des différentes formes de problèmes de partitionnement a été étudiée.

Le problème du partitionnement contraint est celui le plus étudié dans la littérature, il correspond au problème de la répartition de charge dans les machines parallèles. C'est pour résoudre ce type de problème que la plupart des méthodes de partitionnement ont été crées. Cependant, ce problème n'est pas celui du découpage de l'espace aérien. Ce dernier correspond au problème du partitionnement non contraint, qui est le plus souvent résolu grâce à des méthodes spectrales ; mais, comme nous allons le voir, d'autres méthodes classiques de partitionnement peuvent aussi être adaptées au cas non contraint, ainsi que des métaheuristiques.

Première partie

Méthodes de partitionnement de graphe

Chapitre 2

Méthodes classiques de partitionnement de graphe

Ce chapitre présente un état de l'art sur les méthodes de partitionnement de graphe. À la fin de celui-ci, deux nouveaux algorithmes basés sur les méthodes classiques de partitionnement sont introduits.

Ce chapitre commence par résumer les approches qui se sont montrées infructueuses pour le découpage aérien (*infra* 2.1). Puis sont décrites les méthodes d'expansion de région (*infra* 2.2) et la méthode spectrale (*infra* 2.3). L'algorithme d'affinage de Kernighan-Lin (*infra* 2.4) et quelques unes de ces nombreuses améliorations sont ensuite présentés (*infra* 2.5). L'état de l'art se termine sur la présentation de la méthode multi-niveaux (*infra* 2.6). Enfin, l'avant-dernière section de ce chapitre introduit deux nouveaux algorithmes, l'un pour le partitionnement, l'autre pour la répartition de charge (*infra* 2.7). La dernière section de ce chapitre le conclura et parlera de méthodes de partitionnement actuellement prisées (*infra* 2.8).

2.1 Approches infructueuses pour le découpage de l'espace aérien

Plusieurs méthodes déterministes ont été testées au début de cette thèse pour résoudre le problème du partitionnement de graphe non contraint. Cette section présente celles dont l'application au problème du découpage de l'espace aérien s'est révélée infructueuse.

2.1.1 Solution exacte

La première approche naturelle pour résoudre un problème d'optimisation est de parcourir de façon exhaustive l'espace de recherche des solutions admissibles, d'évaluer chacune de ces solutions et de conserver celle(s) dont la valeur associée à la fonction objectif est minimale. Évaluer la pertinence de cette première approche est une étape essentielle avant de se lancer dans l'utilisation de méthodes combinatoires.

Cependant, comme le font remarquer Brian Kernighan et Shen Lin dans [KER70], la recherche exhaustive de la ou les partition(s) de coût minimale est impossible, car l'espace de recherche est beaucoup trop grand. En effet, la taille de celui-ci, c'est-à-dire le nombre de partitions de k parties d'un ensemble à n éléments, est donné par le nombre de Stirling de seconde espèce (*supra* 1.3). Or le nombre de Stirling de seconde espèce évolue de façon exponentielle par rapport à n. On comprend donc pourquoi le partitionnement de graphe fait partie des problèmes d'optimisation combinatoire, et l'inutilité de la recherche exhaustive même pour des graphes de quelques dizaines de sommets.

2.1.2 Flux maximum - coupe minimum

Lester Ford et Delbert Fulkerson sont les créateurs éponymes d'un théorème et d'un algorithme [FOR56]. Ceux-ci sont des outils classiques de résolution de problèmes de réseaux, et plus précisément de problèmes de flux origines-destinations dans un réseau.

Théorème 1 (Flux maximum - coupe minimum) *Soit un graphe orienté $G = (S, A)$, dont les arcs sont pondérés par un poids strictement positif. Le poids maximum du chemin entre deux sommets distincts est égal au coût de coupe minimum d'une coupe séparant ces deux sommets.*

Le principe de l'algorithme de Ford-Fulkerson est de trouver le flux maximum, *i.e.* le chemin de poids maximum, entre deux sommets d'un graphe orienté. D'autres algorithmes basés sur les mêmes principes ont été créés pour trouver une bissection de coupe minimale [NAG94, BRI05], comme l'algorithme *Simple Min-Cut* de Mechthlid Stoer et Frank Wagner [STO97]. Cependant, plusieurs problèmes se posent à leurs adaptations au partitionnement de graphe :
- ces algorithmes n'ont pas de mécanismes pour contraindre la taille des parties de la bissection. Or, comme les bissections trouvées par ces algorithmes sont le plus souvent de tailles très différentes, l'utilisation des ces algorithmes pour le partitionnement de graphe contraint est en pratique impossible ;
- les bissections trouvées ont comme particularité d'être de coût de coupe minimale entre deux sommets distincts. Pour résoudre le problème du partitionnement non contraint, il suffit alors d'appliquer un de ces algorithmes à chaque paire de sommets du graphe. Cela est en pratique irréalisable de par la nature combinatoire d'une telle approche. Il faut donc itérer un certain nombre de fois l'algorithme en sélectionnant aléatoirement un couple de sommets origine-destination, et conserver la meilleure bissection trouvée. Or, la complexité de ces algorithmes de flux maximal est élevée. En effet, à titre d'exemple, la complexité de l'algorithme *Simple Min-Cut* est $O(n_S n_A + n_S^2 log(n_S))$ (où $n_S = card(S)$ et $n_A = card(A)$). Le temps d'exécution d'un tel algorithme de bissection serait donc prohibitif, et son adaptation au k-partitionnement encore davantage.

L'implémentation de l'algorithme *Simple Min-Cut* et son utilisation sur des parties restreintes du problème de découpage de l'espace aérien européen (*infra* 5) nous ont montré que cette approche était inutilisable, tant par la lenteur des algorithmes, que par la très mauvaise qualité des résultats obtenus.

2.1.3 Méthodes de classification

Le but des méthodes de classification est de construire une partition d'un ensemble d'objets dont on connaît les distances deux à deux. La classification a pour hyponyme le partitionnement de données, qui se traduit en anglais par *data clustering*. La classification est une branche très importante de l'informatique, et qui a de nombreuses applications. Les problèmes de classification sont très proches de ceux du partitionnement de graphe.

Il existe de nombreuses méthodes de classification [JAI99, BER02] et quelques articles présentent des rapprochements possibles entre classification et partitionnement de graphe [DHI04c, DHI07]. Les techniques de classification les plus classiques sont :
- la méthode des **centres mobiles** [KAN02]. C'est une méthode itérative qui consiste à calculer pour chaque partie de la partition son centre de gravité, puis à recréer une partition où chaque

partie est constituée des éléments les plus proches de son centre de gravité. Cette méthode rapide et très répandue requiert la notion de distance ;

– les méthodes **hiérarchiques**. Ces méthodes créent des arbres dont chaque niveau constitue une partition. Elles reposent toutes sur le même principe : créer un ensemble de partitions réparties hiérarchiquement en classes de moins en moins fines (*i.e.* en partitions possédant de moins en moins de parties). Chaque nouvelle partition est obtenue par regroupements successifs de parties de la partition directement précédente dans la hiérarchie. Il existe des méthodes hiérarchiques ascendantes ou descendantes. Dans le cas des méthodes ascendantes, les deux individus les plus proches sont regroupés pour former un sommet de l'arbre et ainsi de suite, jusqu'à ce qu'il n'y ait plus qu'un seul individu.

Les méthodes de classification reposent sur la connaissance d'une fonction de distance, ou au minimum d'une fonction de dissimilitude, entre tous les couples d'éléments de l'ensemble à classer. Cependant, il n'existe pas de relation de distance immédiate entre tous les éléments d'un graphe quelconque. Si une relation de distance peut être créée, elle est en général très coûteuse à mettre en place, particulièrement pour les graphes non connexes. Ainsi, les méthodes de classification pour le partitionnement de graphe sont en général inutilisables.

Cependant, dans une certaine mesure, les méthodes hiérarchiques ascendantes pourraient être utilisées sans avoir la connaissance de la distance entre chaque individu. Dans ce cas, elles fonctionneraient de proche en proche à partir des distances connues entre éléments voisins. Dans une telle adaptation, chaque élément serait un sommet du graphe, et la distance entre voisins serait le coût associé à l'arête liant ce sommet à un autre sommet. En réalité, de telles approches par voisinage de sommets existent déjà pour le partitionnement de graphe, ce sont les méthodes d'expansion de région (*infra* 2.2).

2.1.4 Autres approches

La méthode utilisant le nombre de Reynolds[1] que nous avions développée a été adaptée au partitionnement de graphe non contraint. Cette méthode est présentée dans [BIC04a, BIC05]. Cependant, la lenteur de cette méthode et sa difficulté d'adaptation la rendent inutilisable telle quelle aux problèmes de partitionnements de graphe. Pour autant, la méthode reste dans sa structure valable et est proche des méthodes d'expansions de régions. Elle nous a en particulier servi d'inspiration pour la méthode de percolation (*infra* 2.7.1).

2.2 Les méthodes d'expansion de région

Les méthodes d'expansion de région, *graph growing* en anglais, sont des méthodes déterministes. Elles sont très simples à mettre en œuvre et efficaces sur des problèmes de faibles tailles. Leur inconvénient est qu'elles sont de forte complexité temporelle. Les qualités de ces méthodes font qu'elles sont utilisées de manière sous-jacente par les méthodes multi-niveaux, pour créer des partitions à partir de graphes de tailles réduites.

Ces méthodes ne sont pas seulement utilisées dans un rôle de partitionnement, mais peuvent aussi l'être dans un but d'affinage ou de répartition de charge d'une partition [MEY05, PEL07], notamment grâce aux algorithmes de diffusion [HU98].

En tant que méthodes de partitionnement, les méthodes d'expansion de région suivent toutes le même schéma directeur. Partant de plusieurs sommets du graphe qui constituent les prémisses des

[1] Le nombre de Reynolds permet de distinguer les écoulements laminaires des écoulements turbulents

parties de la partition, l'algorithme consiste à rassembler les sommets du graphe dans chacun de ces ensembles par une méthode de propagation directe. C'est-à-dire, un sommet ne peut être ajouté à un de ces ensembles que s'il est adjacent à au moins un des sommets de l'ensemble.

2.2.1 *Graph growing Algorithm* (GGP)

Le *graph growing algorithm* (GGP) a été introduit dans [KAR98a]. Cette méthode de bissection de graphe peut être utilisée comme méthode de partitionnement par un algorithme multi-niveaux.

L'algorithme GGP consiste à créer de manière itérative un ensemble E rassemblant la moitié des sommets du graphe en terme de poids. Cet ensemble E est initialisé en choisissant aléatoirement un sommet dans le graphe. Pendant l'exécution de l'algorithme, les sommets du graphe sont répartis en trois ensembles : l'ensemble E, l'ensemble des sommets adjacents à E, appelé frontière de E et noté F, et l'ensemble des sommets restants, noté R. À chaque itération, les sommets adjacents au(x) sommet(s) de l'ensemble E sont ajoutés à E. Le processus s'arrête lorsque E contient un ensemble de sommets qui représente la moitié du poids total des sommets du graphe.

Si cette méthode est très simple à mettre en œuvre et très rapide en temps d'exécution, la qualité de la partition obtenue dépend principalement du sommet de départ. Parce que, dans le cas de l'utilisation par une méthode multi-niveaux, le nombre de sommets du graphe à partitionner est très petit (souvent moins de 200), cet algorithme peut être exécuté plusieurs fois (une dizaine) avec un sommet de départ toujours différent. La bissection de coût de coupe le plus bas sera alors choisie.

Enfin, la bissection obtenue peut-être avantageusement affinée par un algorithme de Kernighan-Lin, afin de diminuer son coût de coupe.

2.2.2 *Greedy graph growing Algorithm* (GGGP)

L'algorithme GGP a été amélioré par ses auteurs dans [KAR98a] en y ajoutant la notion de gain de coupe d'un sommet due à Kernighan-Lin [KER70]. Cette amélioration a donné naissance au *greedy graph growing algorithm* (GGGP).

L'algorithme GGGP (*cf* algorithme 2) a la même structure itérative que l'algorithme GGP et répartit aussi les sommets du graphes dans les trois ensembles E, F et R décrits précédemment, mais il inclut différemment les sommets dans E. Comme dans l'algorithme de Kernighan-Lin (*infra* 2.4), à chaque sommet s de F, est associé un gain qui correspond à la diminution du coût de coupe qui résulterait de l'insertion de s dans E. Ainsi, les sommets de la frontière de la partition peuvent être triés en fonction de leur gain.

Comme dans le cas de l'algorithme GGP, l'algorithme GGGP commence par initialiser l'ensemble E en y introduisant un sommet tiré aléatoirement dans S. Les ensembles F et R peuvent alors être initialisés à leur tour. Chaque itération de l'algorithme consiste à insérer dans E, le sommet de F permettant de diminuer le plus le coût de coupe. Ce sommet noté s, est le sommet de gain maximal de F. Puis chaque sommet de R adjacent à s est déplacé vers F et son gain est calculé. De même, chaque sommet de F adjacent à s voit son gain recalculé. L'itération suivante peut alors commencer. L'algorithme se termine quand le poids total de E est égal à la moitié de celui de G.

Comme le font remarquer George Vipin Karypis et Kumar, afin de rendre l'algorithme plus rapide, la structure de données Fiduccia-Mattheyses (*infra* 2.5.1) doit être utilisée. L'unique différence est qu'au lieu de calculer le gain pour chaque sommet non bloqué, le gain n'est calculé que pour les sommets de F.

Si l'algorithme GGGP est sensible au choix du sommet initial de E, il l'est moins que pour GGP. Cependant, comme dans le cas de GGP, cet algorithme peut être lancé plusieurs fois (4 dans

Algorithme 2 Algorithme d'expansion de région GGGP.

Procédure GGGP($G = (S, A)$)
  Prendre au hasard une sommet $s_0 \in S$
  $E \rightarrow \{s_0\}$
  $F \rightarrow \{s \in R$ tel que $(s, s_0) \in A\}$
  Calculer les gains de F
  tant que $poids(E) < \frac{1}{2}poids(S)$ **faire**
    Prendre le sommet $s_i \in F$ de gain maximal
    Déplacer s_i de F vers E
    pour toute arête (s, s_i) dans A **faire**
      si $s \in F$ **alors**
        mettre à jours le gain de s
      sinon si $s \notin E$ **alors**
        ajouter s à F
        calculer le gain de s
      fin si
    fin pour
  fin tant que
  retourner E
fin Procédure

[KAR98a]) afin de diminuer l'importance du choix du sommet initial. Les auteurs considèrent que, même dans ce cas, GGGP reste aussi rapide que GGP, tout en produisant des bissections de meilleure qualité.

2.3 La méthode spectrale

2.3.1 Présentation

L'utilisation de la méthode spectrale pour résoudre les problèmes de partitionnement de graphe est très ancienne. Les premiers articles proposant une telle approche sont dus à W. Donath et A. Hoffman [DON72, DON73]. La méthode spectrale a été beaucoup utilisée pour résoudre les problèmes de partitionnement de graphe avant l'arrivée des méthodes multi-niveaux. Son importance était capitale pour le partitionnement de graphe, à tel point que l'un des premiers articles proposant l'utilisation d'une méthode multi-niveaux, n'utilise celle-ci que pour simplifier le calcul nécessaire à la méthode spectrale [BAR93, BAR94]. Par la suite le rapport de force s'est inversé, et c'est la méthode spectrale qui est devenue un des outils des méthodes multi-niveaux [HEN95b].

De nombreux articles présentent l'application de la méthode spectrale au partitionnement de graphe, tant pour le problème du partitionnement contraint [HEN95b, BAR94, HEN93, POT90] que pour celui du partitionnement non contraint [HAG92a, DIN01b, DHI04a, DHI04c].

Les résultats théoriques liés à la méthode spectrale appliquée au partitionnement de graphe sont eux aussi nombreux. Ils permettent de fixer une limite inférieure au coût de coupe d'une partition [DON73, BEZ99, ELS03], ou encore d'évaluer la qualité d'un séparateur[2] du graphe [GUA98]. Dans [ALO85], N. Alon et V. Milman prouvent que l'obtention d'une coupe grâce à la méthode spectrale a

[2]Un séparateur d'un graphe est un sous-ensemble de sommets dont la suppression rend le graphe non connexe.

au pire un coût quadratique de celui de la partition optimale.

La méthode spectrale doit son nom au théorème spectral de l'algèbre linéaire. Ce théorème permet d'affirmer la diagonalisation des matrices symétriques réelles. Il justifie également la décomposition des matrices symétriques réelles en valeurs propres dans une base orthonormale de vecteurs propres. Or, le problème du partitionnement de graphe peut, au prix de deux approximations (*infra* 2.3.5), être ramené à la résolution d'un système numérique $Mx = \lambda x$. Résoudre ce système numérique consiste à trouver une base orthogonale de vecteurs propres de la matrice M.

2.3.2 Quelques résultats d'analyse numérique

Nous allons voir dans cette section comment le problème du partitionnement de graphe peut être résolu en trouvant une famille de valeurs propres et une famille de vecteurs propres d'une matrice. Les résultats présentés dans cette section ont fait l'objet de plusieurs publications, principalement dans le cas de graphes non pondérés [POT90, HAG92a, HEN93]. Cependant, nous généraliserons ces concepts pour les étendre aux cas des graphes valués, comme dans [MOH91, MOH97].

Avant de continuer, nous allons présenter quelques définitions d'analyse numérique qui nous serviront dans cette section.

Définition 21 (Vecteur propre, valeur propre) *Soit E un espace vectoriel sur un corps K. Soit u un endomorphisme de E. Un vecteur x de E tel qu'il existe λ dans K tel que $u(x) = \lambda x$ est appelé vecteur propre de u. Dans ce cas, λ est appelé valeur propre de u.*

Si l'on se ramène à un endomorphisme sur les réels, et que M est la matrice de cet endomorphisme, alors on appelle vecteur propre de M tout vecteur x (*i.e.* matrice colonne) de réels qui vérifie $\exists \lambda \in R, Mx = \lambda x$.

Définition 22 (Matrice semi-définie positive) *Soit M une matrice symétrique réelle d'ordre n. Elle est dite semi-définie positive si elle vérifie l'une des deux propriétés équivalentes suivantes :*

1. *pour toute matrice colonne non nulle x à n éléments réels, on a $x^T M x \geq 0$;*
2. *toutes les valeurs propres de M sont positives ou nulles.*

Considérons la matrice Laplacienne d'un graphe G telle que définie en section 1.2 :

$$M_{Lap} = M_{Deg} - M_{Adj} \ .$$

On a la propriété suivante :

Propriété 1 *La matrice Laplacienne M_{Lap} d'un graphe G non-orienté et positivement pondéré, est semi-définie positive.*

Preuve : Soit x un vecteur de taille n. On a pour tout $i \in \{1, \ldots, n\}$:

$$\begin{aligned}
(x^T M_{Lap})_i &= x_i deg(i) - \sum_{j=1}^{n} x_j p(i,j) \\
&= x_i \sum_{j=1}^{n} p(i,j) - \sum_{j=1}^{n} x_j p(i,j) \\
&= \sum_{j=1}^{n} (x_i - x_j) p(i,j) \ .
\end{aligned}$$

Puis,

$$x^T M_{Lap} x = \sum_{i,j=1}^{n} (x_i^2 - x_i x_j) p(i,j) \ .$$

Or, le graphe est non-orienté, $\forall (i,j) \in 1, \ldots, n^2, p(i,j) = p(j,i)$, d'où :

$$x^T M_{Lap} x = \sum_{(s_i,s_j) \in A} (x_i - x_j)^2 p(i,j) \ . \tag{2.1}$$

Puisque le graphe est positivement pondéré, M_{Lap} est semi-définie positive. □

Considérons que l'on cherche à trouver une bissection de G en deux ensembles de sommets S_1 et S_2. Nous verrons dans la section suivante (*infra* 2.3.3) comment généraliser cette méthode au 2^i-partitionnement.

Soient un graphe $G = (S, A)$ et x un vecteur de taille n tel que :

$$\forall s_i \in S, x_i = \begin{cases} 1 & if\ s_i \in V_1 \ , \\ -1 & if\ s_i \in V_2 \ . \end{cases}$$

Si l'on considère le coût de coupe entre S_1 et S_2 défini en section 1.5 :

$$\begin{aligned} coupe(S_1, S_2) &= \sum_{s_i \in S_1, s_j \in S_2} p(s_i, s_j) \\ &= \frac{1}{2} \sum_{(s_i,s_j) \in A} (x_i - x_j)^2 p(i,j) \ . \end{aligned}$$

Si l'on utilise le résultat de l'équation (2.1), alors :

$$coupe(S_1, S_2) = \frac{1}{2} x^T M_{Lap} x \ .$$

Ainsi, minimiser le coût de coupe d'une bissection revient à trouver un vecteur x qui minimise $x^T M_{Lap} x$.

Écrit autrement, ce problème consiste à résoudre le système linéaire :

$$M_{Lap} x = \lambda x \tag{2.2}$$

en trouvant le plus petit réel λ possible. Comme nous l'avons vu au début de cette section, cela revient à trouver une valeur propre minimale de la matrice Laplacienne M_{Lap} du graphe G, ainsi que le vecteur propre correspondant.

2.3.3 Trouver les valeurs propres de la matrice Laplacienne du graphe

Comme nous l'avons vu dans la section précédente (*supra* 2.3.2), chercher les vecteurs propres de la matrice Laplacienne d'un graphe permet de résoudre un problème de partitionnement. Plusieurs méthodes peuvent être utilisées pour trouver les vecteurs propres d'une matrice :

- l'algorithme itératif de Lanczos ;
- la méthode itérative du quotient de Rayleigh ;
- l'algorithme de la décomposition QR ;
- l'algorithme itératif de Jacobi.

Toutes ces méthodes sont présentées dans [GOL96].

Parmi ces méthodes, c'est l'algorithme itératif de Lanczos qui est le plus souvent utilisé [POT90, SIM91, HAG92a, HEN95b]. Cependant, la méthode itérative du quotient de Rayleigh est aussi utilisée dans certains travaux [BAR94, HEN95c]. Dans [HEN95c] elle est même avantageusement couplée avec une méthode multi-niveaux pour la rendre plus rapide.

D'après la propriété 1, la matrice Laplacienne d'un graphe est semi-définie positive, donc toutes ses valeurs propres sont positives. Ainsi, elles peuvent être classées par ordre croissant : $0 \leq \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. Il existe un vecteur propre trivial : le vecteur unité ($x = (1, \ldots, 1)^T$). Sa valeur propre est $\lambda = 0$; on en déduit que $\lambda_1 = 0$.

La première personne qui a étudié les propriétés de ces vecteurs et valeurs propres est Miroslav Fiedler, qui a présenté ses résultats dans [FIE75]. C'est pourquoi cette famille de vecteurs propres est souvent appelée la famille des vecteurs de Fiedler. Ainsi, le premier vecteur de Fiedler est le vecteur unité. Le second vecteur de Fiedler correspond à la valeur propre non-nulle la plus petite de la famille de valeurs propres. C'est le second vecteur de Fiedler qui va permettre de trouver une bissection du graphe. En effet, si les valeurs de ce vecteur sont réelles, leur discrétisation vers $\{-1; 1\}$ permet de répartir les sommets du graphe dans les deux ensembles S_1 et S_2.

Considérons maintenant que l'on cherche une 2^i-partition de G. Le résultat trouvé dans le cas de la bissection d'un graphe se généralise facilement. En effet, les $i+1$ premiers vecteurs de la famille de Fiedler de la matrice Laplacienne de G permettent de trouver une 2^i-partition de G. Pour cela, il suffit de discrétiser chaque vecteur de Fiedler vers $\{-1; 1\}$, puis de créer les bissections correspondantes. Ce procédé sera étudié plus en détail dans les sous-sections suivantes (*infra* 2.3.5 et 2.3.6).

2.3.4 Borne inférieure pour le partitionnement de graphe contraint

L'étude des propriétés spectrales de la matrice d'adjacence du graphe a permis de trouver une borne inférieure au coût de coupe d'une partition d'un graphe non pondéré. La démonstration en a été faite par W. Donath et A. Hoffman dans [DON73].

Théorème 2 *Soient $G = (S, A)$ un graphe de poids unitaires et M_{Adj} sa matrice d'adjacence.*
Soit M_D une matrice diagonale quelconque telle que $tr(M_D) = \sum_i (M_D)_{ii} = 2n_A$.
Soient $m_1 \geq m_2 \geq \cdots \geq m_k$, k entiers positifs tels que $\sum_i m_i = n_S$.
Soit $P_k = \{S_1, \ldots, S_k\}$, une partition de S en k parties telles que $\forall i \in \{1, \ldots, k\}$, $card(S_1) = m_i$.
Si l'on note les valeurs propres de la matrice $M_D - M_{Adj}$ par ordre de valeurs croissantes :
$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_k \leq \cdots \leq \lambda_{n_S}$, alors :

$$coupe(P_k) \geq \frac{1}{2} \sum_{i=1}^{k} card(S_i)\lambda_i$$

Dans ce théorème, la matrice des degrés du graphe peut être utilisée à la place de M_D. Les valeurs propres utilisées sont alors les k plus petites valeurs propres de la matrice Laplacienne du graphe.

Dans le cas des partitions parfaitement équilibrées, Robert Elsässer, Thomas Lücking et Burkhard Monien montrent dans [ELS03], que lorsque $coupe(P_k) = \frac{1}{2} \sum_i card(S_i)\lambda_i$, alors tous les sommets du graphe sont adjacents à la coupe. La partition a dans ce cas un grand nombre de parties, assez proche du nombre de sommets. Ils proposent une nouvelle borne inférieure pour les partitions n'entrant pas dans ce cadre.

2.3.5 Méthodes spectrales pour le partitionnement contraint

Le problème du partitionnement contraint (*supra* 1.5) consiste à trouver une partition P_k en k parties qui respecte une balance de partitionnement bal_{max} et qui soit de coût de coupe minimum. Les méthodes spectrales ont été utilisées dans de nombreux articles pour résoudre le problème du partitionnement contraint [SIM91, HEN93, BAR94, ALP95b, HEN95b].

Dans les deux sous-sections précédentes (*supra* 2.3.2 et 2.3.3), nous avons vu que minimiser le coût de coupe d'une bissection $P_2 = \{S_1, S_2\}$ était équivalent à trouver le second vecteur de Fiedler de la matrice Laplacienne du graphe. Les autres p vecteurs de Fiedler peuvent être utilisés par ordre croissant pour obtenir une 2^p partition. Cependant, comme Bruce Hendrickson et Robert Leland le font remarquer dans [HEN95b], ce procédé n'est pas extensible à l'infini. Seuls les trois premiers vecteurs de Fiedler (sans compter celui de valeur propre nulle) apportent des résultats satisfaisants. Nous allons maintenant expliquer pourquoi.

Soient $G = (S, A)$ un graphe et M_{Lap} sa matrice Laplacienne. Résoudre le système (2.1) est une approche linéaire pour résoudre le problème discret de la minimisation du coût de coupe. Il y a donc deux différences majeures entre résoudre ce système linéaire et trouver une solution au problème du partitionnement contraint :

- les valeurs du vecteur x dans l'équation (2.1) sont des valeurs réelles et non dans $\{-1; 1\}$;
- la résolution du système linéaire (2.1) ne tient pas compte de la balance de la partition.

Ce sont les deux approximations que nous avions évoquées au début de cette section (*supra* 2.3.1).

Heureusement, ces deux inconvénients associés ensemble permettent de trouver une solution au problème du partitionnement contraint. En effet, la discrétisation du vecteur de Fiedler, x, vers $\{-1; 1\}$ va permettre de respecter la balance de partitionnement bal_{max}. Pour cela, il suffit de trier les composantes de x par valeurs croissantes et de discrétiser la première moitié du vecteur vers -1 et la seconde vers 1. L'indice i du vecteur correspondant à cette première moitié est choisi tel que : $x_i < 0$ et cette moitié a atteint le poids maximal autorisé par la balance ; ou $x_i > 0$ et cette moitié a atteint le poids minimal autorisé par la balance. Ainsi, une bissection du graphe respectant la balance a été trouvée.

Lorsque plusieurs vecteurs de Fiedler sont utilisés pour trouver un 2^p-partitionnement, chaque vecteur est discrétisé comme précédemment pour obtenir une bipartition du graphe, en respectant les modifications imposées par les bissections précédentes au niveau de la balance de partitionnement. Ces modifications étant de plus en plus difficiles à respecter, le nombre de vecteurs de Fiedler qui peuvent être utilisés est limité.

Pour une présentation plus formelle et plus détaillée, le lecteur pourra consulter l'excellent article de Bruce Hendrickson et Robert Leland [HEN95b].

Les procédés que nous venons de présenter ne permettent que de trouver des partitions en 2, 4 ou 8 parties. Afin de trouver une k-partition du graphe, avec k quelconque, la technique de bissection récursive (*supra* 1.7.3) peut être utilisée.

La méthode spectrale est une approche globale pour le partitionnement. Il est donc utile d'appliquer à la partition trouvée un algorithme d'optimisation locale [HEN93]. Ils sont appelés algorithmes d'affinage dans le cas du partitionnement de graphe (*supra* 1.7.2). Les algorithmes utilisés pour affiner les partitions trouvées par les méthodes spectrales sont le plus souvent de type Kernighan-Lin (*infra* 2.4).

2.3.6 Méthodes spectrales pour le partitionnement non contraint

Le partitionnement non contraint nécessite la distinction de deux cas, suivant que l'on cherche à minimiser l'une de ces deux fonctions objectifs :

Le ratio de coupe

La minimisation du ratio de coupe par des méthodes spectrales a été proposée par [POT90, HAG92a, DIN01b]. La première étape de ce processus de minimisation consiste, comme dans le cas contraint, à résoudre un système linéaire. Cependant, le système linéaire (2.2) doit être adapté au ratio de coupe. Le nouveau système linéaire à résoudre est :

$$M_{Lap}x = \lambda M_{Adj}x \ .$$

Une fois les vecteurs de Fiedler trouvés par les méthodes proposées en sous-section 2.3.3, il faut les projeter sur la partition. C'est l'étape qui se différencie le plus du cas contraint. En effet, la contrainte sur la balance de partitionnement n'existe plus. Soit x le second vecteur de Fiedler trouvé, ses composantes sont dans $[-1; 1]$. Puisqu'il n'y a plus de balance de partitionnement à respecter, la projection de x sur la bissection $P_2 = \{S_1, S_2\}$ est très simple : si la composante x_i de x est négative, le sommet s_i ira dans S_1 ; dans le cas inverse, si $x_i > 0$ alors s_i sera ajouté à S_2.

La coupe normalisée

La coupe normalisée est très utilisée pour résoudre des problèmes de segmentation d'image. Dans ce cas, ce problème est souvent résolu par une méthode spectrale [SHI98b, SHI98a, WEI99, SHI00, BEN05]. Le processus de minimisation de la coupe normalisée est identique à un détail près de celui du ratio de coupe que nous venons de présenter ; dans le cas de la coupe normalisée, le système linéaire à résoudre est le suivant :

$$M_{Lap}x = \lambda M_{Deg}x \ .$$

2.3.7 Problèmes et améliorations

L'un des inconvénients des méthodes spectrales est qu'elles utilisent des algorithmes de recherche de vecteurs propres qui sont gourmands en temps de calcul, mais surtout en espace mémoire. Ce défaut est rappelé dans [HEN95a] où Bruce Hendrickson et Robert Leland préconisent l'utilisation de l'algorithme itératif de Lanczos pour les graphes de moins de 10 000 sommets, et l'utilisation de la méthode itérative mêlant multi-niveaux (présenté *infra* 2.6) et quotient de Rayleigh pour les graphes plus grands.

Nous avons évalué la quantité de mémoire utilisée et les temps de calcul nécessaires à l'algorithme de Lanczos et à l'algorithme hybride multi niveaux-quotient de Rayleigh. Le tableau 2.1 présente ces résultats. Ceux-ci ont été obtenus grâce à l'outil de partitionnement Chaco sur un ordinateur fonctionnant avec le système d'exploitation GNU/Linux Debian, muni d'un processeur Intel Pentium IV cadencé à 3GHz, disposant de 512Mo de mémoire vive. Ce tableau nous montre les limites de la méthode de Lanczos, et plus généralement des méthodes spectrales non hybrides, tant en espace mémoire qu'en temps de calcul. En effet, pour les graphes testés, qui ont tous une taille de plus de 10 000 sommets, l'algorithme de Lanczos peut utiliser jusqu'à 360Mo de mémoire, contre au plus 44Mo pour l'algorithme hybride. Le temps de calcul est lui aussi bien plus petit dans le cas de l'algorithme hybride avec au plus 0,54 secondes de calcul contre au plus 4,75 secondes pour l'algorithme de Lanczos. De plus, les coûts obtenus avec la méthode hybride nous montrent que cette méthode permet d'obtenir des résultats de qualité au moins similaire à ceux de l'algorithme de Lanczos.

Le constat que nous venons de présenter explique pourquoi les méthodes multi-niveaux, qui permettent de résoudre des problèmes de plus grande taille et plus rapidement, ont été si rapidement utilisées.

Graphes			Lanczos			Quotient de Rayleigh		
Nom	sommets	arêtes	coût	temps(s)	mém.(ko)	coût	temps(s)	mém.(ko)
bcsstk32	44 609	985 046	7779	2,56	155 136	7647	0,41	27 644
t60k	60 005	89 440	186	1,11	164 624	171	0,20	16 512
wing	62 032	121 544	1324	1,20	136 200	1386	0,32	19 420
brack2	62 631	366 559	857	1,13	114 608	856	0,24	24 476
finan512	74 752	261 120	206	2,68	317 452	498	0,34	28 792
fe_tooth	78 136	452 591	4396	2,39	221 604	4372	0,3	30 104
fe_rotor	99 617	662 431	2420	4,75	360 504	2473	0,37	39 988
598a	110 971	741 934	2513	3,70	183 972	2528	0,54	44 012
fe_ocean	143 437	409 593	2222	2,32	300 688	659	0,43	41 588

TAB. 2.1 – Temps de calcul et mémoire utilisés par le programme Chaco en fonction de la méthode spectrale choisie.

2.4 L'algorithme de Kernighan-Lin

2.4.1 Principe général

L'algorithme de Kernighan-Lin [KER70] permet d'affiner la bissection d'un graphe précédemment obtenu. Il s'agit donc d'un algorithme d'optimisation locale. L'idée maîtresse de cet algorithme est de trouver deux sous-ensembles de sommets de même taille, chacun dans une partie de la bissection, tels que leur échange diminue le coût de coupe de la bissection. Afin de trouver « une partition localement optimale » [KER70], l'algorithme procède ainsi : partant d'une bissection existante, l'algorithme échange successivement deux sous-ensembles de la bissection jusqu'à ce que plus aucun sous-ensemble diminuant le coût de coupe ne puisse être trouvé. La dernière bissection obtenue est donc la bissection de coût de coupe minimal trouvé par l'algorithme.

2.4.2 L'algorithme

Dans un premier temps, nous supposerons que nous cherchons une bissection parfaitement équilibrée (*i.e.* de balance de partitionnement unitaire) et que tous les sommets sont de poids identiques. Nous verrons ensuite comment adapter l'algorithme à des parties de tailles différentes.

Pour créer les deux sous-ensembles de la bissection à échanger, les auteurs introduisent deux notions : le coût extérieur et le coût intérieur. Le coût intérieur $I(s)$ d'un sommet s de la partie S_i de la bissection est défini comme la somme des poids des arcs adjacents à s dont le second sommet est dans S_i :

$$I(s) = \sum_{s' \in S_i} poids(s, s') \ . \tag{2.3}$$

Le coût extérieur $E(s)$ d'un sommet s de S_i se définit comme la somme des poids des arcs entre s et les sommets n'appartenant pas à S_i :

$$E(s) = \sum_{s' \in S - S_i} poids(s, s') \ . \tag{2.4}$$

Soit $D(s)$ la différence entre le coût extérieur et le coût intérieur du sommet s :

$$D(s) = E(s) - I(s) \ . \tag{2.5}$$

Soient deux sommets appartenant chacun à une partie de la bissection, $s_1 \in S_1$ et $s_2 \in S_2$. Le gain de l'échange, g, de s_1 et s_2 est :

$$g(s_1, s_2) = D(s_1) + D(s_2) - 2poids(s_1, s_2) . \tag{2.6}$$

Pour chaque sommet $s \in S$, la différence entre le coût extérieur et le coût intérieur de s, $D(s)$, est calculée. Les valeurs $D(s)$ sont réparties en deux ensembles D_1 et D_2 suivant l'appartenance de s à l'une des deux parties.

L'algorithme de Kernighan-Lin, présenté par l'algorithme 3, est construit autour d'une boucle itérative, appelé par les auteurs *phase 1 optimization* ou « passe », qui permet de chercher deux sous-ensembles de la bissection à échanger. À chaque passe, la construction de ces deux sous-ensembles est faite par itérations successives, en ajoutant à chaque itération un élément à chaque sous-ensemble. Soit l'algorithme à l'itération o. L'algorithme commence par chercher deux sommets appartenant chacun à une partie de la bissection (S_1, S_2), maximisant g (cf. équation (2.6)). Pour ce faire, chaque ensemble D_1 et D_2 est trié :

$$\begin{aligned} D_1 \ &: \ D(s_1, 1) \geq D(s_1, 2) \geq \cdots \geq D(s_1, n) \ , \\ D_2 \ &: \ D(s_2, 1) \geq D(s_2, 2) \geq \cdots \geq D(s_2, n) \ . \end{aligned}$$

Une fois les ensembles triés, uniquement un petit nombre de couples de sommets va être considéré. En effet, si deux couples (s_1, i) et (s_2, j) sont trouvés tels que $D(s_1, i) + D(s_2, j)$ est inférieur ou égal au gain maximum précédemment trouvé, alors il ne peut y avoir une autre paire de sommets $((s_1, k), (s_2, l))$ avec $k \geq i$ et $l \geq j$, de gain supérieur à ce gain maximum. Ainsi, deux sommets $s_1^o \in D_1$ et $s_2^o \in D_2$ de gain g^o sont sélectionnés. Ils sont alors verrouillés, ce qui veut dire qu'ils ne pourront plus être sélectionnés pendant le reste de cette passe. Les valeurs $D(s)$ des ensembles D_1 et D_2 sont recalculées pour les sommets non verrouillés en fonction de la nouvelle bissection $(S_1 - \{s_1^o\}, S_2 - \{s_2^o\})$. Soient D_1' et D_2' les nouveaux ensembles :

$$\begin{aligned} D_1'(s) &= D_1(s) + 2poids(s_1^o, x) - 2poids(s_2^o, x), \quad x \in S_1 - \{s_1^o\} \ , \\ D_2'(s) &= D_2(s) + 2poids(s_2^o, x) - 2poids(s_1^o, x), \quad x \in S_2 - \{s_2^o\} \ . \end{aligned}$$

La procédure est alors itérée en partant de la bissection $(S_1 - \{s_1^o\}, S_2 - \{s_2^o\})$ et des ensembles D_1' et D_2'. L'itération se termine lorsqu'il n'y a plus de sommet à échanger. Comme les parties de la bissection sont de tailles égales, chaque partie est alors vide.

Le gain résultant de l'échange de l'ensemble des sommets $\{s_1^1, \ldots, s_1^i\}$ et $\{s_2^1, \ldots, s_2^i\}$ par la passe est le suivant :

$$G(i) = \sum_{k=1}^{i} g^k \ .$$

On peut noter qu'après la dernière itération, le gain total de l'échange est :

$$\sum_{k=1}^{n_S/2} g^k = 0 \ .$$

À la fin de la passe, si le gain obtenu est strictement positif, alors les deux ensembles de sommets qui maximisent G sont échangés.

L'algorithme de Kernighan-Lin procède par passes successives tant que celles-ci permettent de trouver deux ensembles de sommets à échanger. La bissection finale a pour coût de coupe, le coût de coupe initial moins la somme des gains obtenus par les différentes passes.

Algorithme 3 Algorithme de Kernighan-Lin.

Procédure KL($G = (S, A)$,P_2)
  $coût\_coupe \leftarrow coupe(P_2)$
  répète « passe » % aussi appelé *phase 1 optimization*
    Calculer D_1 et D_2
    pour $o = 1$ **à** $n_S/2$ **faire**
      Choisir s_1^o et s_2^o qui maximisent le gain g
      $E_1(o) \leftarrow s_1^o$, $E_2(o) \leftarrow s_2^o$, $G(o) \leftarrow g$
      Verrouiller s_1^o et s_2^o
      Mettre à jour D_1 et D_2
    fin pour
    Choisir o qui maximise G
    si $G(o) > 0$ **alors**
      $coût\_coupe \leftarrow coût\_coupe + G(o)$
      Échanger $E_1(o)$ et $E_2(o)$ dans P_2
    fin si
  jusqu'à ce que $G(o) > 0$
  retourner (P_2, $coût\_coupe$)
fin Procédure

2.4.3 Améliorations proposées par Brian Kernighan et Shen Lin

Pour trouver une bissection du graphe de coût de coupe encore inférieur, les auteurs conseillent de répéter l'algorithme en l'initialisant avec différentes bissections. Ils proposent aussi une autre approche qui consiste à perturber la solution précédemment trouvée par l'algorithme en échangeant aléatoirement quelques sommets de chaque partie. L'algorithme sera alors appliqué à cette nouvelle bissection.

L'algorithme de Kernighan-Lin est d'autant plus rapide et performant que sa partition initiale est de bonne qualité. En effet, si la partition initiale est de bonne qualité, alors le nombre d'itérations de l'algorithme sera petit car il sera rapidement impossible de trouver des sous-ensembles à échanger. De plus, l'optimisation étant locale, une diminution très importante du coût de coupe est peu probable. Ainsi, pour avoir un coût de coupe minimal, il est préférable de partir d'une bissection de coût de coupe faible. Le choix de la méthode qui génère la bissection initiale est donc très important. Les auteurs ne préconisent pas de méthode spécifique pour créer la bissection initiale. Cependant, elle peut être facilement générée par des algorithmes d'expansion de région (*supra* 2.2). C'est cette solution qui est utilisée par certaines méthodes multi-niveaux, dont [KAR98a].

2.4.4 Complexité

La complexité de l'algorithme de Kernighan-Lin est son principal défaut. La création des coûts extérieurs et des coûts intérieurs a pour complexité $O(n_A^2)$. Le temps mis pour trouver les deux sommets à échanger lors d'une passe est en $O(n_A^2 \; log \; n_A)$. La complexité de l'algorithme de Kernighan-Lin est donc en $O(n_A^2 \; log \; n_A)$. Nous verrons dans les sections suivantes d'autres algorithmes qui permettent de diminuer cette complexité.

Cependant, pour accélérer la sélection des deux sommets à échanger, Brian Kernighan et Shen Lin proposent de ne pas trier les listes D_1 et D_2, mais de chercher la valeur la plus importante de chacune de ces deux listes et d'échanger les sommets correspondant à ces valeurs. Cette recherche étant linéaire (*i.e.* en $O(n_A)$), la complexité de l'algorithme de Kernighan-Lin sera en $O(n_S^2)$. L'inconvénient de cette nouvelle recherche est qu'elle ne permet pas de trouver les deux sommets de gain maximal. En effet, cette recherche ne tient pas compte du poids entre les deux sommets cherchés présents dans la formule 2.6.

2.4.5 Parties de tailles différentes

Considérons le cas d'une bissection en une partie de n_1 éléments et une autre de n_2 éléments, avec $n_1 < n_2$. Le problème est d'affiner la bissection pour en obtenir une qui possédera des parties d'au moins n_1 sommets et d'au plus n_2 sommets. Brian Kernighan et Shen Lin proposent une adaptation de leur algorithme pour un tel problème. Le nombre de paires de sommets qui seront échangées à chaque passe est réduit à n_1. À la partie de taille n_1, $n_2 - n_1$ sommets « virtuels » sont ajoutés. Ces sommets virtuels ont un poids égal aux autres sommets, mais ne sont reliés par aucun arc aux autres sommets. Le coût, comme le gain de transfert d'un de ces sommets d'une partie à l'autre est donc nul. Après que l'algorithme de Kernighan-Lin ait été appliqué, les sommets virtuels sont retirés de la bissection affinée. La bissection obtenue a deux parties d'au moins n_1 sommets et au plus n_2 sommets.

2.4.6 Sommets de poids différents

Jusqu'à présent, seuls des sommets de même poids ont été considérés. Pour supprimer cette restriction, les auteurs proposent de remplacer chaque sommet de poids $p > 1$ par p sommets de poids unitaires connectés entre eux. Le problème de cette approche est qu'elle agrandit fortement la taille du problème et que certains sommets risquent d'être divisés entre les deux parties. Cette solution n'est donc pas viable pour la plupart des problèmes.

2.5 Améliorations de l'algorithme de Kernighan-Lin

De nombreuses améliorations de l'algorithme de Kernighan-Lin ont été proposées. Cet algorithme ayant été créé pour la bissection de circuits VLSI et ceux-ci étant souvent modélisés avec des hypergraphes, de nombreuses améliorations concernent les hypergraphes [FID82, KRI84, SAN89, HAG97, ZHA02, PAP07]. Cependant, on doit à l'un de ces articles, [FID82], l'implémentation systématiquement utilisée pour diminuer la complexité de cet algorithme, tant dans le cas du partitionnement de graphe que d'hypergraphe. Nous verrons dans une première section (*infra* 2.5.1) cette implémentation.

Dans le cas précis du partitionnement de graphe, et indépendamment des articles sur les méthodes multi-niveaux, seuls quelques articles présentent des améliorations de l'algorithme de Kernighan-Lin [BUI89, DUT93, PEL96, RAN01]. Cependant, les articles présentant les méthodes multi-niveaux proposent souvent un nouvel algorithme d'affinage (*cf infra* 2.6.2) de type Kernighan-Lin dans le cas de la bissection [KAR98a] ou du k-partitionnement [HEN95c, WAL97, KAR98d, KAR98c, WAL00a].

2.5.1 L'implémentation de Fiduccia-Mattheyses

Dans [FID82], C. Fiduccia et R. Mattheyses proposent un algorithme d'affinage basé sur l'algorithme de Kernighan-Lin, mais pour le partitionnement d'hypergraphes. S'intéressant toujours à un problème de bissection, ils réutilisent la notion de gain introduite dans [KER70] et présentée en section 2.4. L'intérêt majeur de cet article est la description de leur structure de données servant à maintenir à jour les gains des sommets.

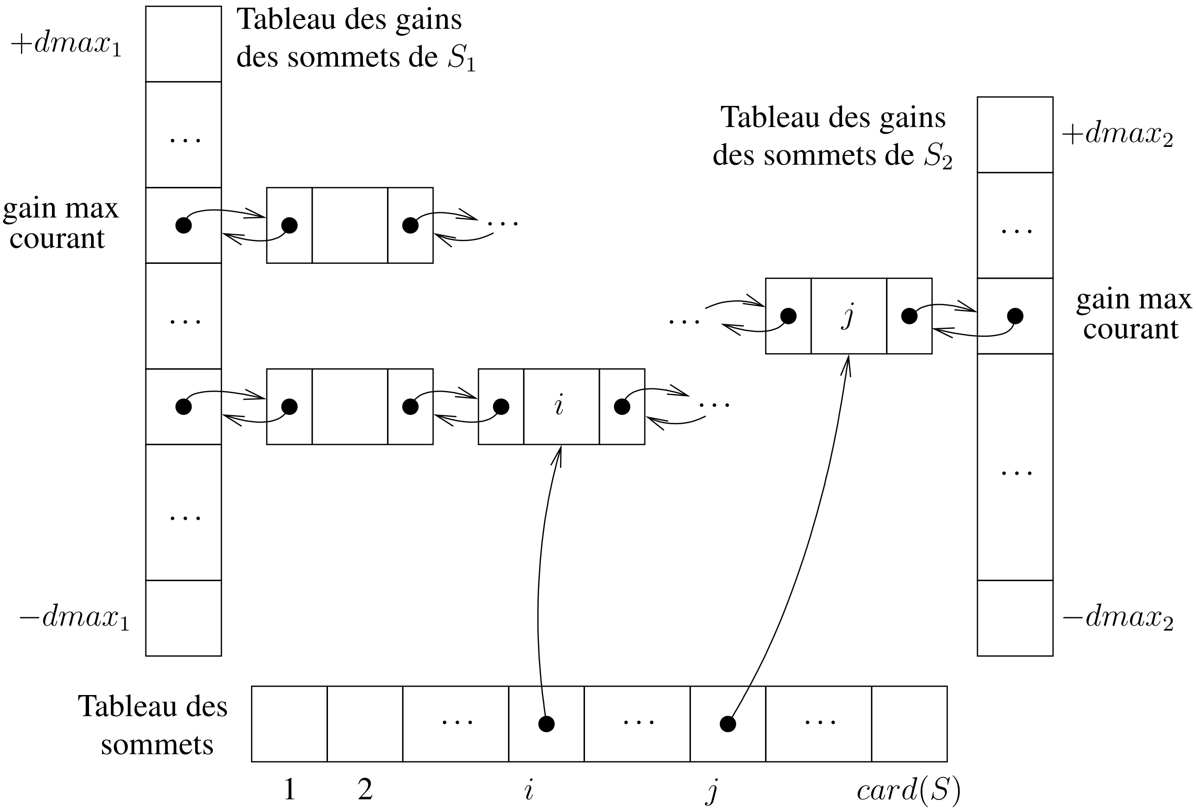


FIG. 2.1 – Implémentation du tableau de gain de Fiduccia-Mattheyses. Tableau des sommets pointant sur des doubles listes chaînées liées au gain du sommet correspondant.

Le gain d'un sommet représente la diminution du coût de coupe engendrée par le déplacement de ce sommet vers l'autre partie de la bissection. Formellement, le gain $g(s)$ d'un sommet s est égal à la différence entre le coût extérieur et le coût intérieur du sommet. Suivant la formule (2.5), $g(s) = D(s)$.

La structure de données qui permet de maintenir à jour le gain des sommets est présentée figure 2.1. Cette structure de données consiste à maintenir pour chaque partie de la bissection un tableau trié des gains des sommets. Une liste doublement chaînée est associée à chacune des cases du tableau de gains. Cette liste doublement chaînée contient les sommets de gain correspondant au rang de la liste dans le tableau de gains. Enfin, chaque case du tableau des sommets pointe vers son sommet, celui-ci appartenant à une des listes doublement chaînées. Ainsi, l'accès à un sommet se fait en temps constant, et son déplacement d'un gain vers un autre gain lors de la mise à jour des gains se fait aussi en temps constant.

Algorithme 4 Algorithme de Fiduccia-Mattheyses adapté au partitionnement de graphe.

Procédure FM($G = (S, A)$, P_2, $balmax$) %P_2 : bissection de G ; $balmax$: balance max de P_2
  $co\hat{u}t\_coupe \leftarrow coupe(P_2)$
  répète « passe » % aussi appelé *phase 1 optimization*
    Créer les deux tableaux de gains
    pour $i = 1$ **à** n_S **faire**
      Choisir $s \in S$ non verrouillé et de gain maximal, respectant après déplacement $balmax$
      pour tout sommet non verrouillé s' adjacent à s **faire**
        Mettre à jour le tableau de gain pour s'
      fin pour
      Verrouiller s
    fin pour
    Choisir i qui maximise le gain G
    si $G(i) > 0$ **alors**
      Modifier la bissection P_2 en fonction de i.
      $co\hat{u}t\_coupe \leftarrow co\hat{u}t\_coupe + G(i)$
    fin si
  jusqu'à ce que $G(i) > 0$
  retourner (P_2, $co\hat{u}t\_coupe$)
fin Procédure

L'algorithme 4 présente l'adaptation de l'algorithme de Fiduccia-Mattheyses au partitionnement de graphe.

L'initialisation du gain pour chaque sommet a pour complexité $O(n_A)$. De même, la mise à jour du gain des sommets étant constant, chaque passe de l'algorithme de Kernighan-Lin se fait en $O(n_A)$. Ainsi, la complexité générale de l'algorithme est linéaire par rapport à la taille de A, ce qui est une grande amélioration par rapport à celle de l'algorithme de Kernighan-Lin, qui était en $O(n_A^2 \; log \; n_A)$.

2.5.2 Adaptation au k-partitionnement

L'algorithme de Kernighan-Lin utilisant l'implémentation de Fiduccia-Mattheyses est efficace et rapide, mais il est prévu pour affiner une bissection. Cependant, dans de nombreux cas la partition à affiner a un nombre $k > 2$ et pas toujours pair de parties. Il est donc utile d'avoir une version efficace de cet algorithme pour le k-partitionnement direct.

Une des premières adaptations de l'algorithme de Kernighan-Lin au k-partitionnement de graphe est due à Laura Sanchis et est décrite dans [SAN89, SAN93]. Cependant, cette adaptation a été faite dans le cadre du partitionnement d'hypergraphes.

Une adaptation de cet algorithme au k-partitionnement est présentée dans [HEN95c] par Bruce Hendrickson et Robert Leland. Cependant, cette adaptation est coûteuse en espace et en temps. En effet, elle utilise $k(k-1)$ tableaux de gains de l'implémentation de Fiduccia-Mattheyses (*supra* 2.5.1) et a une complexité en $O(kn_A)$ à comparer avec la complexité de l'algorithme de Fiduccia-Mattheyses, qui est en $O(n_A)$. Ainsi, cette adaptation n'est utilisable que dans le cas où k est petit.

Pour pallier à ces inconvénients, George Karypis et Vipin Kumar proposent une autre approche dans [KAR98d]. Celle-ci utilise toujours le principe de gain et l'implémentation de Fiducci-Mattheyses, mais a une complexité indépendante de nombre de parties de la partition cherchée. Dans leur algorithme, seul un tableau de gains est utilisé. Cependant, de l'aveu même des auteurs, si leur méthode d'affinage est plus rapide, elle est aussi moins efficace que celle de Bruce Hendrickson et Robert Leland. La section 2.5.3 présente cette méthode de manière plus détaillée.

2.5.3 *Global Kernighan-Lin refinement*

L'algorithme d'affinage *Global Kernighan-Lin refinement* (GKLR) présenté dans [KAR98d] utilise la même structure de données qui permet de maintenir à jour le gain des sommets que celle de l'implémentation de Fiduccia-Mattheyses (*supra* 2.4.2), à la différence qu'un seul tableau de gain est utilisé. Comme tous les algorithmes du type de celui de Kernighan-Lin, cet algorithme est itératif et chaque itération est appelée une « passe » (*supra* 2.4.2). Chaque passe consiste à déplacer un ensemble de sommets vers d'autres parties afin de diminuer le coût de coupe de la partition.

L'algorithme 5 décrit le processus d'affinage du GKLR. Au début d'une passe, le gain de chaque sommet est calculé selon la formule (2.5), mais en ne considérant que deux parties à la fois : celle du sommet et une autre partie adjacente à ce sommet. Dans cet algorithme, le gain d'un sommet est le gain maximum de tous les gains calculés entre ce sommet et les parties adjacentes à ce sommet. Si ce gain est positif, il est conservé, sinon le sommet est verrouillé. Les auteurs ont fait le choix de ne conserver que les sommets dont le gain est positif au début d'une passe, car cela a pour avantage de réduire sensiblement le temps d'une passe. Cependant, cela a aussi pour inconvénient de ne pas pouvoir atteindre certains minima locaux.

La structure de données de Fiduccia-Mattheyses étant constituée des sommets échangeables et de leurs gains correspondants, une itération sur les sommets non verrouillés commence. Chacune de ces itérations commence par la sélection du premier sommet de la liste doublement chaînée de gain le plus important. Soit s ce sommet ; l'algorithme va choisir la partie adjacente à la partie de s qui maximise le gain de s tout en respectant la balance de partitionnement. Ainsi, le sommet s peut être déplacé vers une partie de gain non maximal, et ne correspondant donc pas au gain du tableau des gains. Une fois le sommet s déplacé vers sa nouvelle partie, celui-ci est verrouillé et les gains des sommets adjacents non verrouillés sont mis à jour. À la fin de l'itération, le sommet s est verrouillé. L'itération sur les sommets non verrouillés continue tant qu'il existe des sommets non verrouillés ou qu'un nombre prédéfini, dep, de déplacements successifs de sommets n'a pas conduit à au moins une amélioration du coût de coupe. Pendant l'itération, les gains sont mis à jour et peuvent devenir strictement négatifs. Ainsi, le gain de la partition peut être diminué.

Une fois cette itération terminée, l'ensemble des sommets, s'il existe, ayant permis de décroître le plus le coût de coupe est déplacé vers les parties correspondantes. Le déplacement de ces sommets termine la passe.

L'itération sur les passes s'achève lorsqu'un nombre prédéfini de passes a été exécuté ou lorsqu'une passe n'a pas fait décroître le coût de coupe de la partition.

Algorithme 5 Algorithme d'affinage *Global Kernighan-Lin refinement*.

Procédure GKLR($G = (S, A)$, P_k, $balmax$) %P_k : partition de G ; $balmax$: balance max de P_k
  répète « passe »
    Créer le tableau de gains et l'ensemble des sommets non verrouillés S_v
    tant que $S_v \neq \emptyset$ ou le gain d'un déplacement n'a pas été négatif dep fois de suite **faire**
      Choisir le sommet s de plus grand gain
      pour toute partie $S_i \in P_k - \{S_j, s \in S_j\}$ telle que $S_i \cup \{s\}$ respecte $balmax$ **faire**
        Calculer le gain de l'ajout de s à S_i
      fin pour
      Choisir la partie $S_i \in P_k$ de gain g maximal par rapport au sommet s
      Enregistrer le déplacement de s vers S_i et le gain g correspondant
      pour sommet non verrouillé s' adjacent à s **faire**
        Mettre à jour le tableau de gain pour s'
      fin pour
      Verrouiller s
    fin tant que
    Choisir l'ensemble des sommets à déplacer qui maximise le gain g
    si $g > 0$ **alors**
      Modifier la partition P_k en fonction de l'ensemble des sommets à déplacer
    fin si
  jusqu'à ce que $g > 0$ ou un nombre prédéfini de passes est atteint
  retourner P_k
fin Procédure

L'un des inconvénients majeurs de cet algorithme d'affinage est qu'il a du mal à affiner correctement des partitions qui ne respectent pas déjà la balance de partitionnement. Cet algorithme est souvent utilisé par les méthodes multi-niveaux (*supra* 2.6). Or l'étape de partitionnement des méthodes multi-niveaux ne produit pas toujours une partition respectant la balance de partitionnement.

2.5.4 Algorithme d'affinage de Walshaw-Cross

L'algorithme d'affinage de Walshaw-Cross est présenté dans [WAL00a]. Il s'inspire de nombreuses méthodes différentes. L'algorithme est basé sur l'idée de gain de Kernighan-Lin, et utilise l'implémentation de Fiduccia-Mattheyses pour maintenir les gains à jour. La partie d'affinage proprement dite de cet algorithme est proche de l'algorithme GKLR [KAR98d] (*supra* 2.5.3). Comme dans celui-ci, les sommets sont déplacés un à un et un seul tableau de gain est nécessaire. Cependant, les auteurs ayant remarqué que l'algorithme GKLR ne pouvait rétablir la balance des partitions mal balancées, ils ont utilisé un algorithme de répartition de charge pour équilibrer les parties entre elles. Celui-ci est basé sur une méthode de diffusion décrite dans [HU98]. Cet algorithme ne permet pas de respecter une contrainte de partitionnement, mais a pour but de rééquilibrer parfaitement la partition. Il permet de déterminer quel poids transférer entre les parties du graphe, alors que l'algorithme de type GKLR utilisé après détermine quels sommets transférer.

Les méthodes de diffusion minimisent la norme Euclidienne du poids transféré entre les parties. L'idée sous-jacente de ces méthodes est de trouver quel poids transférer entre des parties adjacentes pour équilibrer la partition. Le transfert de poids ne concerne que les parties adjacentes, ceci afin d'éviter d'augmenter le coût de coupe de la partition. Pour ce faire, l'algorithme de diffusion décrit dans [HU98] cherche pour une partition $P_k = \{S_1, \ldots, S_k\}$, à résoudre le système linéaire :

$$M_{Lap} x = b \ ,$$

où b est un vecteur de taille $k = Card(P_k)$ tel que chacune de ces composantes soit :

$$b_i = poids(S_i) - \frac{1}{k} poids(S) \ ,$$

et où M_{Lap} est la matrice Laplacienne de la partition P_k prise comme un nouveau graphe, avec :

$$(M_{Lap})_{ij} = \begin{cases} poids(S_i) & \text{si } i = j \ , \\ -1 & \text{si } i \neq j \text{ et si } S_i \text{ adjacent à } S_j \ , \\ 0 & \text{sinon .} \end{cases}$$

Le poids à transférer entre les parties S_i et S_j est alors donné par la différence $x_i - x_j$. Il est utile de remarquer que ce poids à transférer est orienté, c'est-à-dire que le poids à transférer entre S_j et S_i est l'opposé du poids à transférer entre S_i et S_j. Pour résoudre ce système linéaire, les auteurs utilisent l'algorithme itératif du gradient conjugué. Celui-ci est facilement initialisable, puisque l'on connaît le premier vecteur propre de M_{Lap} (*supra* 2.3.2), qui est unitaire.

Les auteurs de [HU98] supposent que le graphe est connexe, mais dans le cas contraire, ils proposent d'utiliser l'algorithme présenté dans [SON94] pour recréer ces flux de poids à transférer.

Une fois que l'algorithme de diffusion a trouvé les flux de poids à transférer, un algorithme de type GKLR est utilisé mais avec plusieurs variantes :
- au départ, seuls les sommets sur la frontière de la partition sont rangés dans le tableau des gains. Puis, au fur et à mesure de l'itération, lorsqu'un sommet se retrouve sur la nouvelle frontière, il est ajouté au tableau des gains. Inversement, un sommet n'étant plus sur la frontière est retiré de la liste des gains ;
- le tableau de gains utilisé est trié par gain puis par poids, et chaque combinaison gain/poids est liée à une double liste chaînée. Si le gain est positif, alors les sommets sont ordonnés par ordre de poids croissant, sinon, ils sont ordonnés par poids décroissant. En effet, il est préférable de transférer 3 sommets de poids 1 et de gain 1 qu'un sommet de poids 3 et de gain 1. Inversement, il est préférable de déplacer un sommet de poids 3 et de gain -1 que 3 sommets de poids 1 et de gain -1 ;
- un déplacement de sommet ne tient pas compte du gain. Si le poids de la partie ajouté à celui du sommet, respecte la balance de partitionnement, alors le déplacement est accepté, sinon, si le déplacement du sommet réduit le flux de poids à transférer, alors il est aussi accepté ;
- à la fin de chaque itération, et non à la fin de chaque passe, la nouvelle partition affinée P_r est comparée à la meilleure partition P précédemment obtenue et est sauvegardée à la place de cette dernière si elle remplit une des conditions suivantes : P ne respecte pas la balance de partitionnement et $bal(P_r) < bal(P)$; ou $coupe(P_r) < coupe(P)$; ou $coupe(P_r) = coupe(P)$ et $bal(P_r) < bal(P)$;
- les sommets adjacents à plusieurs parties peuvent être sélectionnés plusieurs fois. Supposons un sommet adjacent à S_i et S_j tel que ses gains soient respectivement $g_i > g_j$. Le sommet est initialement placé dans le tableau des gains au rang g_i. Si le déplacement de ce sommet vers S_i n'est pas accepté, alors le sommet est déplacé au rang g_j, puis lorsque ce sommet sera de nouveau sélectionné, il sera testé pour un déplacement vers S_i puis vers S_j. Si aucun des deux déplacements n'est accepté, alors le sommet sera verrouillé et il sera supprimé du tableau de gains.

L'idée de n'utiliser dans un premier temps que les sommets aux frontières au lieu de tous les sommets a été introduite dans [HEN95c], puis plus précisément dans [WAL95]. Cette idée est extensible au cas des sommets à une certaine distance de la frontière, avec pour distance la notion de plus court chemin entre deux sommets. Une autre extension serait de limiter le parcours de l'algorithme de Kernighan-Lin à ces seuls sommets. En effet, dans le cas des algorithmes multi-niveaux, la partition à affiner est globalement de bonne qualité comme cela est rappelé dans [KAR95]. La partition affinée a donc de grandes chances d'être très proche de la partition originale. Ainsi, cela diminuerait le temps d'exécution de l'algorithme. De plus, cette technique permettrait de conserver la structure « globalement bonne » de la partition (c'est-à-dire dans un puits énergétique), pendant l'étape d'affinage et de projection de l'algorithme multi-niveaux. En effet, un affinage trop poussé de la partition alors que celle-ci n'est pas encore projetée sur le graphe d'origine, risque de la faire sortir du puits énergétique vers une solution localement meilleure à ce niveau de projection, mais globalement moins bonne pour sa projection sur le graphe d'origine.

2.6 La méthode multi-niveaux

2.6.1 Introduction

La méthode multi-niveaux, *multilevel* en anglais, dérive des méthodes *multi-grid* utilisées en physique depuis les années 1970. Cette méthode a été parallèlement utilisée pour la première fois par Stephen Barnard et Horst Simon [BAR93] et par Thang Bui et Curt Jones [BUI93]. Cependant, c'est grâce aux travaux de Bruce Hendrickson et Robert Leland [HEN95c] et de George Karypis et Vipin Kumar [KAR95] que la méthode multi-niveaux est devenue populaire pour le partitionnement de graphe.

Parce que la méthode multi-niveaux est très performante pour résoudre des problèmes de partitionnement de graphe, elle a progressivement remplacé l'utilisation des méthodes spectrales dans la seconde moitié des années 1990.

Jusqu'au début des années 2000, l'application de la méthode multi-niveaux à l'optimisation combinatoire est restée cantonnée aux problèmes de partitionnement de graphe et plus précisément de partitionnement contraint. C'est principalement à Chris Walshaw que l'on doit l'adaptation de cette méthode à d'autres problèmes d'optimisation combinatoire [WAL04], comme le problème du voyageur de commerce [WAL01b, WAL02b], de la coloration de graphe [WAL01a], de la représentation graphique de graphes [WAL03] ou de la tournée de véhicules [ROD05].

Des algorithmes hybrides réunissant méthodes multi-niveaux et métaheuristiques sont apparus très rapidement. L'un des premiers travaux proposant une telle approche est dû à Charles Alpert, Jen-Hsin Hagen et Andrew Kahng [ALP96b]. Leur algorithme est basé sur une méthode évolutionnaire qui utilise un algorithme multi-niveaux.

Dans le domaine du partitionnement de graphe, ce sont les méthodes hybrides qui permettent de trouver les partitions de coûts minimaux. Ainsi, les partitions de coûts minimaux de la *Graph Partitioning Archive* de Chris Walshaw (présentée en section A.3), ont été pour la plupart trouvées par deux méthodes hybrides. La première est une recherche locale itérative (ILS) sur un algorithme multi-niveaux [WAL04] et la seconde un algorithme génétique modifiant la pondération du graphe en fonction des partitions trouvées par un algorithme multi-niveaux [SOP04a].

2.6.2 Principe de la méthode multi-niveaux

La méthode multi-niveaux est actuellement la méthode la plus utilisée pour résoudre des problèmes de partitionnement de graphe contraint. La littérature concernant cette méthode est abondante. Initialement, cette méthode a été créée pour accélérer les outils de partitionnement de graphe existants [BAR93]. Cependant, elle a vite été reconnue comme étant une méthode très efficace et qui apportait une vision plus globale du graphe [KAR98a] que les méthodes d'expansion de région ou d'affinage classiques. Ainsi, la méthode multi-niveaux s'est imposée comme une stratégie « globale » utilisant des algorithmes locaux de partitionnement de graphe.

La méthode multi-niveaux cherche à répondre à une question simple : comment créer rapidement une partition d'un graphe G de grande taille, sachant qu'il est très coûteux de s'occuper de chaque sommet un par un ? La réponse naturelle à cette question est de regrouper les sommets ensemble pour s'occuper de groupes de sommets plutôt que de sommets indépendants. C'est cette idée qui est à l'origine de la méthode multi-niveaux.

La méthode multi-niveaux se décompose en trois phases bien distinctes qui agissent chacune sur le graphe G.

- **Contraction :** la phase de contraction est de nature itérative. À chaque itération, les sommets du graphe résultant de l'itération précédente sont regroupés pour former un graphe similaire, mais dont le nombre de sommets est plus petit que le précédent. Ainsi, une famille de graphes $\{G_1, \dots, G_n\}$, avec $G_1 = G$ est créée, telle que pour chaque graphe G_{i+1}, chacun de ses sommets représente un groupe de sommets du graphe G_i précédent. Le processus s'arrête lorsque le graphe est suffisamment petit ou lorsque le nouveau graphe engendré a une taille trop proche du graphe précédent.
- **Partitionnement :** le but de cet étape est de créer une partition P_k^n du graphe G_n. Pour cela, le graphe G_n résultant de l'étape de contraction peut être partitionné en utilisant une heuristique de partitionnement comme par exemple une méthode d'expansion de région.
- **Affinage :** l'étape d'affinage consiste à projeter le découpage de la partition P_k^n sur le graphe initial G. Cependant, projeter directement P_k^n sur G produit souvent un découpage de qualité médiocre, car localement peu optimal. Pour résoudre ce problème, une famille de partitions $\{P_k^1, \dots, P_k^n\}$ est créée dont chaque élément P_k^i correspond au partitionnement de P_k^{i+1} projeté sur G_i. Pour améliorer localement la partition finale, chaque partition intermédiaire P_k^i est affinée en utilisant un algorithme local d'affinage de type Kernighan-Lin ou *helpfull sets*. La partition finale $P_k = P_k^1$ conjugue alors les propriétés d'être globalement et localement de bonne qualité.

Le schéma de la méthode multi-niveaux est présenté sur la figure 2.2. Les trois phases de la méthode y sont décrites. L'étape de contraction de la méthode multi-niveaux va permettre d'obtenir un aperçu global du graphe. Ainsi, même si la phase de partitionnement utilise un algorithme local de partitionnement, la partition résultante sera globalement de bonne qualité pour le graphe d'origine, mais certainement localement mauvaise. La phase d'affinage va permettre de l'améliorer localement, tout en conservant la structure globalement bonne de la partition.

L'algorithme 6 présente l'application de la méthode multi-niveaux au partitionnement de graphe. Cet algorithme reste très général, et son efficacité dépend fortement des méthodes utilisées par les fonctions « contraction », « partitionner » et « affinage ». De nombreuses méthodes sont utilisées pour réaliser ces trois fonctions. L'énumération de ces différentes méthodes est l'objet des trois prochaines sections.

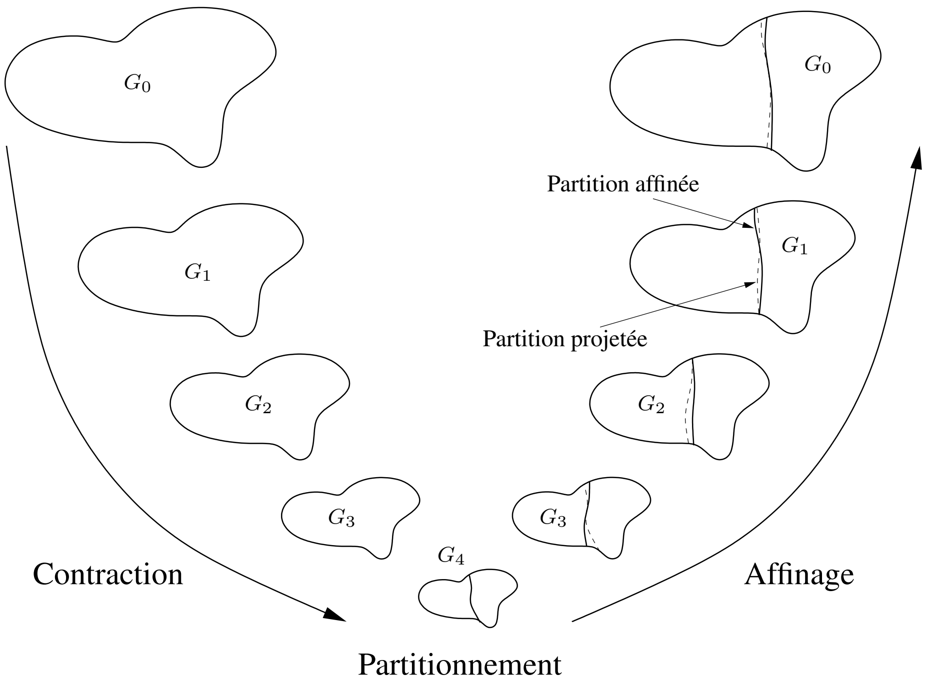


FIG. 2.2 – Les trois phases de l'algorithme multi-niveaux.

Algorithme 6 Algorithme multi-niveaux pour le partitionnement de graphe.

Procédure MULTILEVEL(G,k)
- $G_1 = G$
- $i = 1$
- **répète**
  - $i \leftarrow i + 1$
  - G_i = contraction(G_{i-1})
- **jusqu'à ce que** contraction faible ou G_{i+1} suffisamment petit
- P_k^i = partitionner(G_i,k)
- **pour** $j = i - 1$ **à** 1 **faire**
  - P_k = projection(P_k^{j+1},G_j)
  - P_k^j = affinage(P_k)
- **fin pour**
- **retourner** P_k^1

fin Procédure

2.6.3 Contraction

L'étape de contraction est une étape fondamentale du processus multi-niveaux. En effet, si le regroupement de sommets est de mauvaise qualité, quelle que soit la qualité de l'outil de partitionnement utilisé, la partition résultante sera de mauvaise qualité.

Comme nous l'avons vu dans la section précédente, pendant la phase de contraction, une famille de graphes $\{G_1, \ldots, G_n\}$, avec $G_1 = G$, est créée. Cette contraction peut être faite de différentes manières, comme nous allons le voir.

D'une manière générale, la phase de contraction consiste à regrouper un ensemble de sommets du graphe G_i pour former un unique sommet dans le graphe G_{i+1}. Si l'on note s le sommet résultat dans G_{i+1}, alors le sous-ensemble de sommets de G_i qui l'engendrent est noté S_i^s. Pour pouvoir respecter l'équilibrage des partitions entre les différents graphes G_j, le poids du sommet $s \in G_{i+1}$ sera égal à la somme des poids des sommets de S_i^s. Les arêtes du graphe G_{i+1} seront créées afin de préserver les informations de connectivité entre les graphes, mais aussi afin de préserver le coût de coupe entre les parties des partitions correspondant aux différents graphes G_j. Ainsi, le poids de l'arête (s_1, s_2) sera égal à la somme des poids des arêtes liants $S_i^{s_1}$ à $S_i^{s_2}$. Ainsi, le coût de coupe et balance de partitionnement d'une même partition seront préservés entre les différents graphes G_j.

L'objectif principal du partitionnement de graphe étant de minimiser le coût de coupe entre les parties, l'approche la plus intuitive pour agréger les sommets consiste à regrouper ensemble les sommets liés par les arêtes dont le poids est maximal. Ainsi, les arêtes de poids faible devraient avoir le plus de chances d'être coupées lors de l'étape de partitionnement. Deux possibilités se présentent quant à la taille des ensembles S_i^s créés : si chaque ensemble S_i^s possède au plus deux sommets, alors on parlera de *matching* ; si le nombre de sommets de ces ensembles peut-être supérieur à 2, alors on parlera de *clustering*.

La contraction par *clustering* a été utilisée pour le partitionnement relaxé de circuits VLSI dans [HAG92b]. Elle est particulièrement appréciée dans le cas du partitionnement relaxé, car le poids des arêtes intérieures à chaque partie cherche à être maximisé, comme c'est le cas pour le coût normalisé ainsi que pour le ratio de coût. Au contraire, dans le cas du partitionnement de graphe contraint, le *matching* est plus souvent utilisé.

La technique de contraction par *matching* consiste à trouver un appariement maximal (*supra* 1.2) de chaque graphe G_j et à regrouper les paires de sommets de l'appariement [BUI93, HEN95c]. Un appariement maximum pourrait contenir plus d'arêtes, cependant il mettrait trop de temps à être calculé par rapport au gain obtenu [KAR98a]. Parce qu'un appariement maximal est utilisé, le nombre de sommets de G_{i+1} ne peut pas être inférieur à la moitié de celui de G_i. Ainsi, il faudra au moins $O(log(n_S/n'))$ contractions pour obtenir un graphe de n' sommets. Cependant, en fonction de la connectivité des sommets de G_i, l'appariement maximal peut être beaucoup plus petit que $card(S_i)/2$. Dans ce cas, le rapport des tailles des ensembles de sommets entre les deux graphes, $card(S_i)/card(S_{i+1})$, peut être bien inférieur à deux. Dans le cas où la diminution de taille entre deux graphes deviendrait trop petite, ce processus de contraction est stoppé. Cependant le dernier graphe contracté, G_n, possède en général peu de sommets par rapport au graphe initial.

L'algorithme le plus simple pour créer un appariement maximal est un algorithme itératif aléatoire. A chaque itération, une arête est prise au hasard dans le graphe $G_i = (S_i, A_i)$ et en est retirée, ainsi que toutes ses arêtes adjacentes. Le sommet formé de la réunion des deux sommets de l'arête choisie est ajouté à G_{i+1}, ainsi que les arêtes correspondantes à celle retirées de G_i. La complexité d'un tel algorithme est en $O(card(A_i))$.

L'algorithme 7 présente la méthode de contraction introduite dans [HEN95c] pour le logiciel Chaco. Son principe est de prendre de manière aléatoire une arête (s_1, s_2) du graphe G_i, d'agréger ses deux sommets pour former un sommet du nouveau graphe G_{i+1}, puis de supprimer dans G_i chaque arête adjacente à s_1 et à s_2 pour créer les arêtes correspondantes dans G_{i+1}. Il en résulte un appariement maximal du graphe. La complexité de cet algorithme est en $O(card(A_i))$.

Algorithme 7 Version étendue de l'algorithme de contraction de sommets de Hendrickson-Leland.

```
Procédure CONTRACTION(G_i = (S_i, A_i))
    A ← A_i
    A_{i+1} ← A_i
    S_{i+1} ← S_i
    tant que A ≠ ∅ faire
        Choisir au hasard une arête (s_1, s_2) dans A
        Supprimer l'arête (s_1, s_2) de A et de A_{i+1}
        Supprimer s_1 et s_2 de S_{i+1}
        Ajouter le nouveau sommet s = {s_1, s_2} à S_{I+1}
        poids(s) = poids(s_1) + poids(s_2)
        pour tout (s_1, u) ∈ A faire
            si (s_2, u) ∈ A alors
                poids(s_1, u) ← poids(s_1, u) + poids(s_2, u)
                Supprimer (s_2, u) de A_{i+1}
            fin si
        fin pour
    fin tant que
    retourner G_{i+1} = (S_{i+1}, A_{i+1})
fin Procédure
```

Une autre méthode de contraction, basée sur la recherche d'un appariement maximal, mais utilisant les poids des arêtes du graphe, est présentée dans [KAR98a]. Il est utilisé dans l'outil de partitionnement Metis. Les auteurs ont appelé cette méthode « Heavy Edge Matching » (HEM). Le but de celle-ci est de trouver un appariement maximal du graphe qui minimise le coût de coupe. Pour $G_i = (S_i, A_i)$, si l'on note M_i l'appariement de G_i qui permet de créer G_{i+1}, alors la somme des poids des arêtes de G_{i+1} sera égale à celle de G_{i+1} moins la somme des poids des arêtes de M_i :

$$p(A_{i+1}) = p(A_i) - p(M_i) \ .$$

Or, plus le poids des arêtes du graphe est faible, plus le coût de coupe est petit [KAR95]. Ainsi, il est intéressant de trouver un appariement maximal dont les arêtes sont de poids maximum. La méthode HEM consiste à sélectionner aléatoirement un sommet parmi les sommets restant du graphe G_i, puis à sélectionner l'arête adjacente de poids maximal. Les deux sommets de cette arête forment un nouveau sommet de G_{i+1}. Il faut ajouter ensuite les arêtes adjacentes à ces deux sommets dans A_{i+1}. L'algorithme 8 présente le pseudo-code correspondant. La complexité de cet algorithme est en $O(card(A_i))$. La méthode HEM est utilisée par d'autres outils de partitionnement de graphe comme Scotch ou Jostle. Bien qu'étant présente dans la bibliothèque Metis, la méthode HEM n'y est pas utilisée par défaut. C'est une version légèrement différente, nommée *Sorted Heavy Edge Matching* (SHEM) qui est utilisée [KAR98b]. La méthode SHEM trie les sommets en fonction de leurs degrés (*supra* 1.2) avant d'appliquer la méthode HEM sur cette liste triée.

Algorithme 8 Algorithme de contraction de sommets HEM.

Procédure CONTRACTIONHEM($G_i = (S_i, A_i)$)
 $S \leftarrow S_i$
 $A_{i+1} \leftarrow A_i$
 $S_{i+1} \leftarrow S_i$
 tant que $S \neq \emptyset$ **faire**
  Tirer au hasard une sommet $s_1 \in S_i$
  Prendre l'arête $(s_1, s_2) \in A_i$ de poids maximal
  Supprimer s_1 et s_2 de S
  Ajouter le nouveau sommet $s = \{s_1, s_2\}$ à S_{i+1}
  $poids(s) = poids(s_1) + poids(s_2)$
  pour tout arête (s_1, u) dans A **faire**
   si $(s_2, u) \in A$ **alors**
    $poids(s_1, u) \leftarrow poids(s_1, u) + poids(s_2, u)$
    Supprimer (s_2, u) de A_{i+1}
   fin si
  fin pour
 fin tant que
 retourner $G_{i+1} = (S_{i+1}, A_{i+1})$
fin Procédure

2.6.4 Partitionnement

La phase de partitionnement de la méthode multi-niveaux a pour but de trouver une partition du graphe contracté. Celle-ci sera la partition initiale de la phase d'affinage.

La solution la plus simple pour créer la partition initiale du graphe contracté est de contracter le graphe jusqu'à ce que son nombre de sommets soit égal à K, c'est-à-dire au nombre de parties de la partition cherchée. Ainsi, le graphe contracté constitue en lui-même la partition initiale. Cette approche est celle de plusieurs travaux : [GUP97, WAL00b, WAL00a].

Il existe plusieurs inconvénients à cette approche. Le principal inconvénient est que la phase d'agrégation produit un graphe dont les sommets sont pour la plupart de poids très différents [WAL00a]. L'étape d'affinage devra donc être très performante pour rétablir la balance de partitionnement de la partition initiale. Pour contrer ce problème, M. Walshaw dans [WAL00b] propose une méthode de contraction qui permet d'obtenir des sommets de tailles similaires, alors que M. Gupta dans [GUP97] propose de successivement désagréger puis agréger les sommets pour rétablir la balance de partitionnement. Un autre inconvénient, qui rejoint souvent le premier, apparaît lorsque le graphe G n'est pas connexe. Dans ce cas, la phase de contraction peut soit aboutir à une partition très mal répartie, soit à un nombre de sommets plus grand que le cardinal de la partition cherchée. Cependant, malgré les limites que nous venons d'énoncer, cette approche a montré son efficacité [WAL00a].

La plupart des algorithmes multi-niveaux utilisent une méthode de partitionnement de graphe déjà éprouvée pour créer la partition initiale. Nous pouvons regrouper ces méthodes en trois catégories :
- **les méthodes spectrales**, présentées en section 2.3 ;
- **les méthodes de contraction**, présentées en section 2.2 ;
- les méthodes gloutonnes.

Les méthodes gloutonnes sont en général peu utilisées pour résoudre des problèmes combinatoires, sauf de très faible taille, car elle ont un temps d'exécution prohibitif. Cependant, une méthode gloutonne est utilisée dans [RON05], et comme les résultats obtenus par l'algorithme multi-niveaux de cet article sont assez bons, il nous a semblé important d'évoquer l'existence de cette approche. Celle-ci consiste à calculer le coût de coupe de chaque bissection du graphe contracté respectant la balance de partitionnement, et à sélectionner la bissection de coût minimal. Cependant, cette méthode est par nature extrêmement coûteuse et donc peu conseillée même si le graphe contracté est très petit (20 sommets dans [RON05], correspondant à 524 287 bissections au total).

La méthode de partitionnement peut soit créer une bissection, soit créer une partition en k parties. Dans ce dernier cas, on dira que l'on utilise en fait du k-partitionnement direct. Cette approche est utilisée dans de nombreux travaux [BAÑ03, WAL00a, KAR98d, HEN95b]. Cependant, de nombreux algorithmes de partitionnement et d'affinage ont été créés pour la bissection de graphe [ALP97, KAR98a, RON05]. Dans ce cas, pour obtenir une partition en k parties, il faut recourir à la technique de la bissection récursive (*supra* 1.7.3). Le choix entre k-partitionnement et bissection récursive se pose pour les phases de partitionnement et d'affinage, mais pas pour la phase de contraction, à la fin de laquelle aucune partition n'est réalisée.

2.6.5 Affinage

La phase d'affinage de la méthode multi-niveaux consiste à projeter sur le graphe initial la partition du graphe contracté trouvée pendant la phase de partitionnement. Cette phase serait triviale si une projection directe de la partition du graphe contracté sur le graphe initial conduisait à une partition localement optimale. Cependant, comme George Karypis et Vipin Kumar le constatent dans [KAR95], une partition localement optimale d'un graphe contracté G_{i+1} à partir de G_i n'est pas forcément aussi bonne pour le graphe G_i.

Des algorithmes d'affinage sont donc utilisés pour améliorer localement le coût de coupe de la partition. Concrètement, puisque la phase de contraction a construit une famille de graphes contractés $\{G_1, \ldots, G_n\}$, la phase d'affinage va consister à récursivement projeter la partition d'un graphe contracté G_{i+1} sur son père, G_i, puis à appliquer sur cette partition un algorithme d'affinage. Ainsi, une famille de partitions $\{P_k^1, \ldots, P_k^n\}$ est créée, dont chaque élément P_k^i correspond au partitionnement de P_k^{i+1} projeté sur G_i. L'itération se termine lorsque la partition P_k^2 projetée sur le graphe initial G_1 est affinée. La partition P_k^1 résultante associe alors une configuration globalement bonne et un découpage localement performant.

De nombreuses méthodes d'affinage existent. Cependant, tous les algorithmes d'affinage ont en commun deux particularités : leur recherche doit être locale, et doit partir d'une partition existante. Beaucoup de ces méthodes sont basées sur l'algorithme de Kernighan-Lin [KER70]. Cependant, d'autres méthodes existent, comme la méthode des « ensembles utiles », en anglais *Helpful-Sets*, présentée dans [DIE95, DIE96, SCH03, MON04]. Le recuit simulé (*infra* 3.2) a aussi été utilisé comme méthode d'affinage dans [MAR96, DIE96, BAÑ03]. Récemment, de nouvelles méthodes d'affinage basées sur les méthodes de diffusion ont vu le jour [PEL07] (*supra* 2.7.1).

Comme le précisent Bruce Hendrickson et Robert Leland dans [HEN95b], l'algorithme d'affinage utilisé par les méthodes multi-niveaux doit être rapide et efficace. Or l'algorithme de Kernighan-Lin présenté en section 2.4, réunit ces deux critères. De plus, l'algorithme de Kernighan-Lin est d'autant plus efficace que la partition initiale qu'il doit affiner est de bonne qualité, ce qui est le cas pour les méthodes multi-niveaux.

Pour améliorer la rapidité de l'algorithme de Kernighan-Lin, l'implémentation décrite par C. Fiduccia et R. Mattheyses dans [FID82] est systématiquement utilisée. Cette implémentation est décrite en sous-section 2.5.1.

Cependant, ces deux méthodes ont été créées pour la bissection de graphe. L'adaptation de ces méthodes au k-partitionnement est présentée section 2.5. Nous avons utilisé comme méthodes d'affinage en plus de celle présentée par C. Fiduccia et R. Mattheyses [FID82], celle de Bruce Hendrickson et Robert Leland [HEN95b], l'algorithme GKLR de George Karypis et Vipin Kumar [KAR98d], ainsi que la méthode de Walshaw et Cross [WAL00a].

2.7 Nouveaux algorithmes issus des méthodes classiques

Cette section présente deux algorithmes que nous avons développés en nous inspirant des méthodes classiques de partitionnement de graphe.

2.7.1 La percolation

Dans cette sous-section, la méthode de percolation va être adaptée au k-partitionnement de graphe. Cette méthode a été conçue pour résoudre le problème du découpage de l'espace aérien, dans le but de servir d'initialisation à différentes métaheuristiques (*infra* 3). Par la suite elle a aussi servi comme méthode locale de bissection à l'algorithme de fusion-fission pour le partitionnement non contraint (*infra* 4.3). Les principes d'écoulement d'un fluide sont utilisés par d'autres méthodes pour l'affinage de partitions ou la répartition de charge [DIE98, MEY05, MEY06, PEL07].

Au sens littéral, la percolation est l'écoulement d'un fluide à travers un solide poreux, tel qu'un filtre. Par exemple, la boisson du café est extraite des graines de café moulues par le processus de percolation. Ce sont les mathématiciens S.R. Broadbent et J.M. Hammersley qui ont introduit le terme de percolation dans [BRO57]. La percolation est un processus de propagation aléatoire d'un fluide à travers un milieu. Le modèle de percolation se définit par le mouvement déterministe d'un fluide à travers un milieu dont la structure est aléatoire. L'aléa ne se trouve pas dans le mouvement du fluide mais dans le milieu à travers lequel il évolue [PAJ01].

La percolation fonctionne par écoulement d'un fluide de proche en proche, le fluide part d'un endroit initial pour se répandre aux alentours. Adapté au k-partitionnement, le fluide devra se répandre à travers les mailles du graphe en s'écoulant à partir de k sommets initiaux. Ces k sommets constituent les prémisses des k parties de la partition finale. Afin de bien distinguer les parties les unes des autres, de chaque sommet initial s'écoulera un fluide de couleur différente. En se répandant de proche en proche à travers le graphe, le fluide passe sur des sommets et leur apporte sa couleur. L'intensité du fluide diminue proportionnellement à la distance parcourue depuis le sommet initial, et inversement proportionnellement à la somme du poids des arêtes du chemin parcouru. Le processus se complexifie quand les fluides se rejoignent. Les fluides ne sont pas mélangeables, c'est-à-dire qu'un fluide ne peut pas pénétrer l'espace d'un autre fluide, il ne peut que le repousser pour récupérer des sommets. Sur un sommet en compétition entre plusieurs fluides, le fluide d'intensité la plus grande l'emporte.

Plus formellement, soient $G = (S, A)$ un graphe et $P_k = \{S_1, \ldots, S_k\}$ un ensemble de k ensembles vides S_i qui deviendra la partition du graphe. L'algorithme 9 présente la méthode décrite ci-après. La partition est initialisée en incorporant un sommet s_i^* différent dans chacune de ces k parties. Puis un sommet connexe à une partie sera ajouté à celle-ci en fonction d'un critère de choix déterministe reposant sur le chemin déjà parcouru par le fluide et le maximum du poids entre ce sommet et chacun des sommets de cette partie. Ainsi, entre le sommet initial et un sommet de la même partie, le flux est égal à la somme du poids des arêtes traversées pondérée par leur distance respective

Algorithme 9 Algorithme de percolation.

Procédure PERCOLATION($G = (S, A)$, $\{s_1^*, \ldots, s_k^*\}$)
  $S_{visités} \leftarrow \{s_1^*, \ldots, s_k^*\}$
  $P_k \leftarrow \{S_1 = \{s_1^*\}, \ldots, S_k = \{s_1^*\}\}$
  $bool \leftarrow vrai$
  mem est initialisé comme étant un tableau de valeurs nulles
  tant que $S_{visités} \neq S$ ou $bool$ **faire**
    $bool \leftarrow faux$
    pour $i = 1$ **à** k **faire** % Boucle sur les parties
      pour tout $s \in S_i$ **faire** % Boucle sur les sommets de la partie S_i
        pour tout s' adjacent à s **faire** % Boucle sur les sommets adjacents du sommets s
          $flux \leftarrow poids(s, s') + \frac{mem(s')}{2}$
          si $flux > mem(s)$ **alors** % Le flux est plus fort avec la partie de s'
            $mem(s) \leftarrow flux$
            si $s' \in S_{visités}$ **alors** % s' était déjà dans une partie
              déplacer s' vers S_i
              $bool \leftarrow vrai$
            sinon
              ajouter s' à $S_{visités}$
            fin si
          fin si
        fin pour
      fin pour
    fin pour
  fin tant que
  retourner P_k
fin Procédure

en nombre d'arêtes : soit $C(s, s_i^*)$ le chemin de s vers s_i^*,

$$C(s, s_i^*) = \{arête \in A \text{ ayant permis d'aller de } s_i^* \text{ à } s\} ,$$
$$\text{et } flux(s, S_i) = \sum_{arête \in C(s,s_i^*)} \frac{poids(arête)}{2^{card(C(s,s_i^*))-1}} .$$

Dans la formule précédente, la division par $2^{card(C(s,s_i^*))}$ permet de tenir compte de l'éloignement entre s et s_i^*. À chaque sommet s appartenant déjà à une partie, est associé le flux le plus important étant arrivé jusqu'à lui dans le tableau mem.

Cette méthode procède par incorporation successive des sommets dans les parties. A chaque itération, tous les sommets d'une partie se voient appliquer cette méthode, ainsi un sommet appartenant déjà à une partie peut malgré tout changer de partie. L'itération s'arrête lorsque tous les sommets sont répartis dans les différentes parties et qu'aucun déplacement de sommet vers une autre partie n'est possible.

Dans cet algorithme, lors de la boucle sur les sommets de S_i, seul compte le poids du flux le plus fort qui atteint s et pas les contributions des arêtes adjacentes à s. Pour un graphe pondéré avec des valeurs peu différentes, cela pourrait être un handicap. Cependant, dans le cas du problème de découpage de l'espace aérien, le graphe est construit autour des flux d'avions les plus importants (de poids largement supérieur aux autres) et ce sont donc eux qu'il ne faut pas couper.

La complexité de l'algorithme au niveau de la boucle sur les parties est en $O(n_A)$. Cependant, cette boucle va être itérée de nombreuses fois et ce nombre d'itérations dépend de la structure du graphe. Le nombre d'itérations est infini si le graphe n'est pas connexe. Cet algorithme permet de créer une partition dont les parties sont connexes, et qui minimise la fonction de coupe normalisée (*supra* 1.4). En effet, la partition créée maximise le poids des flux à l'intérieur de chaque partie, tout en minimisant les flux entre parties.

2.7.2 Algorithme de répartition de charge

Les différentes améliorations de l'algorithme de Kernighan-Lin que nous venons de voir sont toutes des méthodes d'affinage. Cependant, il nous est apparu lors de nos différents tests utilisant des méthodes multi-niveaux, ou la fusion-fission (*infra* 4), que les partitions engendrées ne respectaient pas la balance de partitionnement, et dans certain cas de beaucoup. Si bien que les algorithmes d'affinages n'arrivaient pas à trouver une partition la respectant. C'est à cette occasion que nous nous sommes intéressés aux algorithmes de répartition de charge.

Un tel algorithme est utilisé par la méthode d'affinage de Walshaw-Cross (*supra* 2.5.4). Les algorithmes de répartition de charge utilisent souvent des mécanismes de diffusion [SCH97, HU98, SCH01, PEL07]. Notre algorithme utilise la notion de gain introduite par Brian Kernighan et Shen Lin (*supra* 2.4). Pour limiter le temps d'exécution de l'algorithme, seuls les secteurs appartenant à la frontière de la partition peuvent être déplacés. De même, pour accélérer l'algorithme, l'implémentation de Fiducci-Mattheyses est utilisée (*supra* 2.5.1).

Notre méthode repose sur les principes issus de l'exemple suivant qui pourrait en fait être un « jeu » de patience. Le dispositif de cet exemple est présenté figure 2.3, et illustre les principes régissant le problème de la répartition de charge dans un graphe. Soit un plan carré posé horizontalement et dont le centre est axé sur une rotule à doigt[3] dont les deux axes sont AC et BD. Au milieu

[3]Parmi les liaisons simples en mécanique, la rotule à doigt se différencie de la rotule car elle possède un degré de liberté de moins. C'est-à-dire, elle permet des rotations autour de deux axes de l'espace, alors que la rotule en permet trois.

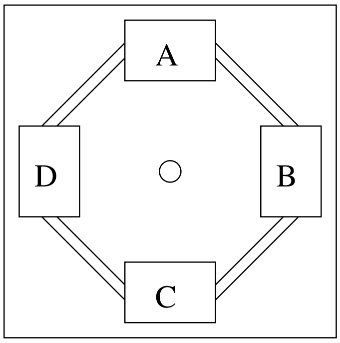


FIG. 2.3 – Exemple illustrant les principes régissant le problème de la répartition de charge dans un graphe.

de chacun des côtés de ce plan sont disposées des boîtes contenant des billes, notées A, B, C et D. Chacune de ces boîtes est reliée aux deux boîtes de ses cotés perpendiculaires par des tubes. Les billes peuvent circuler dans ces tubes. Au début du jeu, le plan est penché sur le coté C. Le but du jeu est d'arriver à faire tenir le plan horizontalement.

Pour pouvoir réaliser cet objectif, il faut que chaque boîte contienne la même quantité de billes. Les règles sont les suivantes : seule une bille peut être déplacée à la fois, et toujours en empruntant un tube ; il y a $4 * moyen$ billes ; le plan s'affaisse du coté de la boîte contenant $max + 2$ billes ($max >= moyen$), et lorsque cela se produit, les jeu est à recommencer. Au début du jeu, il n'y a pas de billes dans C, et plus de $moyen$ billes dans B et D. Pour réussir à mettre le plan à l'horizontal, il suffit de déplacer progressivement et parallèlement les billes de A vers C en les faisant transiter par B et D. Ce qui veut dire qu'il faut faire remonter la pente aux billes et qu'elles vont passer par des boîtes contenant plus que le nombre moyen de billes. Il faut aussi faire en sorte que les billes de B et D ne tombent pas dans A.

Tous ces problèmes sont transposables à ceux du partitionnement de graphe. Si une partie est trop grosse, c'est en général qu'elle a attiré à elle trop de sommets et qu'elle peut donc avoir tendance à en attirer encore plus. Un algorithme de répartition de charge va donc probablement devoir faire des déplacements préjudiciables au coût de coupe de la partition. De plus, si une partie trop grosse est entourée de partie plus petites, mais elles aussi trop grosses, il faut qu'elle déplace certains de ses sommets vers ces parties sans pour autant qu'elle ne devienne moins grosse que les autres, ce qui risquerait de déséquilibrer encore plus la partition. Enfin, une fois que cette partie trop grosse a été délestée de ces sommets, il ne faut pas qu'elle puisse accepter de nouveaux sommets. Ceci dans le but de ne pas tourner en rond. Cependant, si entre temps elle est passée sous le seuil fixé par la balance de partitionnement, elle pourra de nouveau accepter des sommets. Toutes ces conditions nécessitent l'utilisation d'un mécanisme fin de blocage des parties qui les empêchent de reprendre des sommets. C'est le principe des vases communiquants.

L'algorithme 10 présente notre méthode de répartition de charge. Comme la majorité des méthodes basées sur la notion de gain de l'algorithme de Kernighan-Lin. Ce mécanisme de répartition de charge s'articule autour d'une boucle externe effectuant plusieurs « passes ». Le nombre de passages dans cette boucle est limité à $passesmax$. Chacun de ces passages se fait en deux étapes :

- la première étape consiste à créer un tableau de gains pour chaque partie dont le poids dépasse la balance autorisée ; ce poids à ne pas dépasser est noté $poids\_max$. Chaque sommet adjacent à une de ces parties est ajouté dans ce tableau de gains. Soit un sommet s adjacent à la partie S_j

Algorithme 10 Algorithme de répartition de charge.

Procédure REPCHARGE($G = (S, A)$, P_k, $balmax$, $passesmax$)
  $nb\_passes \leftarrow 0$
  répète passe
    $nb\_passes \leftarrow nb\_passes + 1$
    $poids\_max \leftarrow balmax * \frac{poids(S)}{k}$ % poids maximal d'une partie
    Créer k tableaux de gains
    pour toute partie $S_i \in P_k$ telle que $poids(S_i) > poids\_max$ **faire**
      pour toutes les arêtes $(s, s') \in A$ telles que $s \in S_i$ et $s' \notin S_i$ **faire**
        $gain \leftarrow$ gain du déplacement de s vers la partie de s'
        Ajouter $gain$ au tableau de gains de la partie de s'
      fin pour
    fin pour
    Débloquer toutes les parties
    tant que $bal(P_k) > balmax$ **faire** % boucle sur la taille des parties
      pour toute partie $S_i \in P_k$ non bloquée **faire**
        Choisir le sommet s' de plus grand gain
        si gain s' > gain s **alors**
          $s \leftarrow s'$ et $S_{destination} \leftarrow S_i$
        fin si
      fin pour
      si s est vide, *i.e.* aucun sommet s' n'a été trouvé **alors**
        Sortir de la boucle sur la taille des parties
      fin si
      $S_{origine} \leftarrow$ partie de s
      si $poids(S_{origine}) > poids\_max$ et $poids(S_{origine}) > poids(S_{destination})$ **alors**
        $poids\_dest \leftarrow poids(S_{destination})$
        Déplacer le sommet s de $S_{origine}$ vers $S_{destination}$
        si $poids\_dest \leq poids\_max$ et $poids(S_{destination}) > poids\_max$ **alors**
          Bloquer la partie $S_{destination}$
        fin si
        si $poids(S_{origine}) \leq poids\_max$ **alors**
          Débloquer la partie $S_{origine}$
        fin si
      fin si
      Supprimer s de tous les tableaux de gains existants
    fin tant que
  jusqu'à ce que $bal(P_k) \leq balmax$ ou aucune partie n'a changé de poids ou $nb\_passe \geq passesmax$
  retourner P_k
fin Procédure

qui ne respecte pas la balance de partitionnement. On note S_i la partie de s. Le gain correspondant au déplacement de s dans S_j est ajouté au tableau de gains de S_j. Ainsi, seuls les sommets sur la frontière de la partition peuvent faire partie d'un tableau de gains. Une fois les tableaux de gains créés, toutes les parties sont débloquées. Une partie sera par la suite bloquée dans le cas où, étant trop grosse, elle perd un sommet et respecte alors la balance. Ce mécanisme permet d'éviter de sélectionner pour le déplacement d'un sommet, une partie déjà trop grosse.

- la seconde étape a pour but de diminuer la taille de toutes les parties trop grosses existantes ou amenées à l'être. Pour cela elle utilise une boucle interne qui ne se termine que dans deux cas : la partition respecte la balance ou les tableaux de gains des parties non bloquées sont vides. Cette boucle interne se décompose elle-même en plusieurs étapes :
  - dans un premier temps, un sommet est sélectionné dans le but d'un déplacement vers une partie non bloquée. Ce sommet est choisi comme étant celui de gain le plus grand parmi les tableaux de gains des parties non bloquées. Comme nous l'avons dit, si aucun sommet n'a pu être choisi, la boucle interne se termine. Le sommet sélectionné est noté s, la partie à laquelle il appartient $S_{origine}$ est la partie pour laquelle son gain a été choisi $S_{destination}$;
  - dans le cas où le poids de $S_{origine}$ est supérieur à la fois au poids maximum et au poids de $S_{destination}$, alors une série de mises à jour commence :
    - le sommet est réellement déplacé de $S_{origine}$ vers $S_{destination}$ et le poids des parties est recalculé ;
    - si la partie $S_{destination}$ devient trop grosse à cause de l'ajout du sommet s, alors elle est bloquée afin qu'elle ne puisse plus accueillir d'autres sommets ;
    - si le poids de $S_{origine}$ est inférieur au poids maximum $poids\_max$, alors la partie $S_{origine}$ est débloquée (cependant, elle n'était pas obligatoirement bloquée). Elle est débloquée afin qu'à une prochaine itération, elle puisse servir de vase communiquant entre les partitions.
  - enfin, le sommet s est supprimé de tous les tableaux de gains afin qu'il ne puisse plus être déplacé. Cette opération a pour but d'éviter de déplacer loin de leur partie d'origine des sommets possédant peu d'arêtes ou de trop faibles poids.

Le fait de n'utiliser pour la répartition de charge que des sommets à la frontière de la partition a pour inconvénient de fortement pénaliser le coût de coupe dans le cas où la balance de partitionnement est beaucoup trop grande. Cependant, cela permet aussi de réduire le temps d'exécution de l'algorithme. De plus, l'utilisation d'une méthode d'affinage permet par la suite de régler cet inconvénient.

2.8 Conclusion

Ce chapitre a débuté par l'énumération de quelques méthodes qui se sont révélées infructueuses pour le problème du découpage de l'espace aérien : solution exacte, flux maximum, classification. Puis deux techniques de partitionnement ont été étudiées, l'expansion de région et la méthode spectrale. L'étude de celles-ci est suivie par la présentation d'algorithmes d'affinage des partitions ; ainsi, ont été décrits l'algorithme de Kernighan-Lin et plusieurs de ses perfectionnements. La méthode multi-niveaux, qui peut utiliser à la fois les méthodes de partitionnement et d'affinage précédemment décrits, a alors pu être présentée. Enfin, ce chapitre se termine par l'introduction de deux nouveaux algorithmes que nous avons conçu. Le premier algorithme, qui a pour objet le partitionnement de graphe, est basé sur la mécanique des fluides et la percolation. Le second, qui est un algorithme de répartition de charge, est basé sur la notion de gain de Kernighan-Lin.

Actuellement, les outils de partitionnement les plus efficaces utilisent tous la méthode multi-niveaux. Cependant, cette méthode dépend étroitement des performances des algorithmes d'affinage et de partitionnement qu'elle utilise ; l'étude de ceux-ci est donc essentielle. Si la méthode multi-niveaux est depuis quelques années beaucoup plus utilisée que la méthode spectrale, cette dernière continue a être utilisée pour résoudre les problèmes de partitionnement non contraints.

Récemment, de nouvelle méthodes basées sur des analogies avec les flux de liquides, comme la diffusion ou la percolation, ont vu le jour. L'algorithme de partitionnement que nous avons proposé est basé sur l'un des ces mécanismes, la percolation. D'autre part, ayant constaté l'importance de la répartition de charge dans les problèmes de partitionnement de graphe, nous avons cherché un nouvel algorithme répondant à ce problème. L'algorithme de répartition que nous avons introduit utilise la notion de gain, largement utilisée, de Kernighan-Lin. Le nouvel algorithme de partitionnement introduit sera utilisé pour le problème du découpage aérien et celui de répartition de charge, pour les bancs de tests du partitionnement contraint.

Chapitre 3

Différentes métaheuristiques appliquées au partitionnement de graphe

Ce chapitre a pour objet de présenter, d'une part, les différentes adaptations que nous avons faites, de trois métaheuristiques classiques, pour les problèmes du partitionnement de graphe contraint et non contraint, d'autre part, les adaptations actuellement les plus performantes de métaheuristiques pour le partitionnement.

La première section de ce chapitre présentera d'une manière générale ce que sont les métaheuristiques (*infra* 3.1). Les trois sections qui suivront étudieront chacune, dans un premier temps, une métaheuristique particulière, puis dans un second temps, l'adaptation que nous proposons de celle-ci au partitionnement de graphe. Nous proposerons ainsi des adaptations du recuit simulé (*infra* 3.2), des algorithmes de colonies de fourmis (*infra* 3.3) et des algorithmes évolutionnaires (*infra* 3.4), aux problèmes du partitionnement de graphe. La cinquième section de ce chapitre énumérera les adaptations actuellement les plus performantes de métaheuristiques pour le partitionnement de graphe contraint (*infra* 3.5). Enfin, la dernière section récapitulera le travail présenté (*infra* 3.6).

3.1 Introduction générale sur les métaheuristiques

Les métaheuristiques sont apparues dans les années 1980 pour résoudre des problèmes d'optimisation difficile [DRÉ03]. Ces méthodes ont en commun certaines caractéristiques :

- leur nature stochastique leur permet d'explorer plus facilement un espace des solutions de très grande taille ;
- le calcul du gradient de la fonction objectif n'est pas nécessaire à ces méthodes ;
- elles sont inspirées par des analogies avec la nature ;
- un réglage long de leurs paramètres est souvent nécessaire pour obtenir des résultats performants ;
- elles sont coûteuses en temps de calcul.

Le principe de base d'une métaheuristique est de parcourir l'espace des solutions à la recherche de son minimum global en utilisant des mécanismes lui permettant de s'extraire des minima locaux du paysage énergétique. L'un des atouts majeurs des métaheuristiques est leur facilité d'adaptation à de nouvelles fonctions objectifs.

Lorsque le problème d'optimisation possède peu de contraintes, la fonction objectif utilisée par les métaheuristiques est souvent celle du problème. Cependant, quand le problème possède plusieurs contraintes ou que certaines sont non linéaires, la fonction objectif doit prendre en compte la violation des contraintes en pénalisant les solutions les enfreignant. Une autre façon, de plus en plus populaire, de résoudre ce type de problème est d'utiliser une approche multiobjectif.

Les métaheuristiques peuvent être classées en deux grandes familles [DRÉ03], celles de *voisinage* et celles à base de *population*. Les métaheuristiques de voisinage ont en commun d'avoir un mécanisme d'acceptation de solutions dégradées leur permettant de s'extraire des minima locaux. Celles à base de population disposent d'un mécanisme collectif pour se sortir des minima locaux.

Ce chapitre présente trois métaheuristiques classiques, une de voisinage, le recuit simulé (*infra* 3.2), et deux à base de population, les algorithmes de colonies de fourmis (*infra* 3.3) et les algorithmes évolutionnaires (*infra* 3.4).

La méthode multi-niveaux reste encore presque exclusivement appliquée au partitionnement. Pourtant des travaux ont montré qu'elle pouvait être utilisée à la façon d'une métaheuristique sur d'autres problèmes [WAL04], comme le problème du voyageur de commerce [WAL01b, WAL02b] ou celui du coloriage de graphe [WAL01a]. Cependant, comme nous le verrons (*infra* 3.5), les adaptations les plus performantes de métaheuristiques au problème du partitionnement de graphe contraint sont souvent des méthodes hybrides utilisant une métaheuristiques classique avec un algorithme multi-niveaux.

3.2 Le recuit simulé

Le recuit simulé est une métaheuristique s'inspirant de la métallurgie créée au début des années 1980. C'est une méthode bien éprouvée. Elle a été présentée pour la première fois par S. Kirkpatrick, C. Gelatt et M. Vecchi dans [KIR83].

Cette section est divisée en deux sous-sections. La première (3.2.1) présente la méthode du recuit simulé. La seconde (3.2.2) décrit nos deux adaptations de cette méthode au problème du partitionnement de graphe non contraint.

3.2.1 Description de l'algorithme du recuit simulé

La métaheuristique du recuit simulé s'inspire de la technique expérimentale du recuit utilisée en métallurgie. Celle-ci permet d'obtenir un état stable, ou d'énergie minimale, du métal. Cet état est obtenu quand le matériau a trouvé une structure cristalline. Alors que la technique bien connue de la trempe fige le matériau dans un état méta stable, la technique du recuit permet d'éviter au matériau d'être dans un état méta stable caractérisant un minimum local d'énergie.

Pour éviter au matériau de rester bloqué dans un état méta stable, la technique du recuit consiste à porter le matériau à haute température, puis à abaisser lentement celle-ci. En informatique, la méthode du recuit simulé transpose cette technique à la résolution d'un problème d'optimisation. Ainsi, la température, paramètre essentiel du recuit physique, est directement utilisée dans la méthode du recuit simulé comme paramètre de contrôle. De même, par analogie avec le processus physique, l'énergie du système devient la fonction de coût à minimiser. Pour simuler l'évolution thermodynamique du système, le recuit simulé utilise l'algorithme de Metropolis [MET53]. Ce dernier s'appuie sur la distribution de Boltzmann utilisée en physique pour déterminer la distribution des particules en fonction de leur niveau d'énergie.

L'algorithme du recuit simulé est décrit dans de nombreux ouvrages [SIA89, DRÉ03]. Son principe est simple : partant d'un état initial quelconque, aussi appelé configuration initiale, un nouvel état est créé à partir de l'état précédent par une modification élémentaire de cet état. Il est accepté si son énergie est plus faible, sinon il est accepté avec une certaine probabilité. Ce schéma est réitéré en remplaçant l'état initial par l'état accepté tant qu'un »équilibre thermodynamique« n'est pas atteint.

Lorsque cet équilibre est atteint, la température est diminuée. Puis les équilibres thermodynamiques sont renouvelés tant que le système n'est pas figé. Il existe différentes variantes de cette méthode [ING89].

Cette méthode permet donc d'accepter un état dégradé par rapport à l'état précédent. Cette particularité, commune à toutes les métaheuristiques, permet d'explorer une plus grande partie de l'espace des solutions et d'éviter les minima locaux.

Algorithme 11 Algorithme du recuit simulé.

Procédure RECUITGÉNÉRIQUE($e_{init}, T_{max}, T_{min}$)
  $T \leftarrow T_{max}$
  $e \leftarrow e_{init}$
  $e^* \leftarrow e_{init}$
  tant que $T > T_{min}$ **faire** % Boucle de décroissance de la température
    répète % Boucle de stabilisation
      $e' \leftarrow$ modification élémentaire de e
      $\Delta e \leftarrow énergie(e') - énergie(e)$
      si $\Delta e \leq 0$ **alors** % Règle d'acceptation de Metropolis
        $e \leftarrow e'$
      sinon si $proba\_Boltzmann(\Delta e) >$ nombre aléatoire $x \in [0; 1]$ **alors**
        $e \leftarrow e'$
      fin si
      si $énergie(e^*) > énergie(e)$ **alors**
        $e^* \leftarrow e$
      fin si
    jusqu'à ce que le système soit figé
    $T \leftarrow abaisser(T)$
  fin tant que
  retourner e^*
fin Procédure

L'algorithme 11 présente la méthode du recuit simulé pour la minimisation d'un problème d'optimisation combinatoire (*supra* 1.1) de fonction objectif $énergie$ et d'espace des solutions E. L'algorithme prend trois arguments : l'état initial, e_{init}, la température initiale du système, T_{max}, ainsi que sa température finale, t_{min}, qui est la température du système figé. L'état initial e_{init} peut être choisi aléatoirement dans l'espace des solutions, ou être le résultat d'une autre heuristique. Cette dernière solution n'apporte pas toujours un gain de rapidité et d'efficacité, surtout si l'heuristique a trouvé un minimum local duquel le recuit aura du mal à sortir.

L'algorithme du recuit simulé s'articule autour de deux boucles. La première est la boucle de décroissance de la température qui, lorsque le système est figé, décroît la température pour simuler le recuit. La seconde est la boucle de stabilisation qui itère tant que le système n'est pas figé. Les causes de stabilisation du système sont très variées selon les algorithmes de recuit simulé. Dans de nombreux cas, le système est dit figé lorsqu'un nombre prédéfini de modifications élémentaires consécutives a été rejeté par la règle d'acceptation de Metropolis.

La première boucle fait intervenir la boucle de stabilisation, ainsi que la fonction $abaisser$, chargée de diminuer la température. De nombreuses lois de décroissance de la température peuvent être imaginées. Après en avoir testé plusieurs pour notre problème de découpage de l'espace aérien, la loi de décroissance linéaire qui est la plus répandue et aussi la plus simple, est apparue comme étant le choix le plus judicieux. Ainsi :

$$abaisser(T) = T_{fact}T \text{ avec } T_{fact} < 1 .$$

Le paramètre T_{fact} est le plus souvent compris entre $0,9$ et $0,99$.

La boucle de stabilisation crée un nouvel état par modification élémentaire du précédent, puis lui applique la règle d'acceptation de Metropolis. La fonction $proba\_Boltzmann : R \longrightarrow [0;1]$ peut en théorie représenter n'importe quelle densité de probabilité d'une loi de probabilité. Cependant, pour rester fidèle aux origines de la méthode et par simplicité de calcul, le facteur de Boltzmann diminué de la constante de Boltzmann est presque toujours utilisé pour cette fonction :

$$proba\_Boltzmann(\Delta e) = e^{-\frac{\Delta e}{T}} .$$

Ainsi, lorsque la différence d'énergie est positive, plus celle-ci sera faible, plus grande seront les chances que le nouvel état soit accepté. La température joue aussi un rôle important dans cette probabilité d'acceptation. Plus elle sera élevée, et plus les chances de garder le nouvel état le seront aussi.

Le meilleur état trouvé par l'algorithme est conservé et successivement mis à jour par la boucle de stabilisation. C'est cet état qui est retourné quand la première boucle, et donc le recuit, s'achèvent.

Pour que l'algorithme du recuit simulé soit efficace, il faut qu'une modification élémentaire de e créé un état e' très proche de e. Dans ce but, la modification élémentaire de e doit être locale et unique. Dans le cas où le nouvel élément e' serait trop éloigné de e dans l'espace des solutions, alors le recuit se comporterait simplement comme un algorithme de recherche aléatoire.

3.2.2 Notre adaptation du recuit simulé au partitionnement de graphe

Dans la littérature, le recuit simulé a aussi bien été utilisé comme outil de partitionnement de graphe [JOH89, WIL91] que comme outil d'affinage d'une partition [MAR96, DIE96, BAÑ03, GIL06]. Cependant, notre adpatation du recuit simulé a été réalisé exclusivement pour le partitionnement de graphe et pas pour l'affinage.

Nous avons réalisé deux adaptations de la méthode du recuit simulé. Ces deux adaptations résolvent le problème du partitionnement non contraint (*supra* 1.6), afin de pouvoir être utilisé pour le découpage de l'espace aérien européen.

Première adaptation

Nous avons décrit notre première adaptation du recuit simulé au partitionnement non contraint dans [BIC04b]. Dans cet article, l'algorithme est directement décrit pour résoudre le problème de découpage de l'espace aérien. Dans ce paragraphe, il sera présenté comme un algorithme pour le partitionnement non contraint en k parties.

L'algorithme 12 présente cette adaptation. Très fidèle à la méthode générale du recuit simulé, cette implémentation est cependant originale en un point : la modification élémentaire d'un état diffère selon la température du système. Une modification élémentaire d'une partition est facile à trouver : il suffit de déplacer un sommet vers une autre partie. Lorsque la température est élevée ($T > T_{seuil}$), le recuit est utilisé pour s'éloigner des minima locaux et l'entropie du système doit être grande. Dans ce cas, le sommet sera déplacé vers la partie de plus basse énergie (*supra* 1.4) de la partition. Ceci afin de ne pas trop déséquilibrer la partition tout en conservant le caractère très aléatoire du recuit à haute température. Par contre, lorsque la température est basse ($T \leq T_{seuil}$), peu de mouvements sont acceptés. Comme un mouvement faisant perdre la connexité des parties n'a aucune chance d'être accepté, la nouvelle partie du sommet sera choisie parmi les parties connexes à celle du sommet.

Algorithme 12 Première adaptation du recuit simulé.

Procédure RECUIT1($P_k, T_{max}, T_{min}, T_{seuil}, min\_états\_refusés$)
  $T \leftarrow T_{max}$
  $P_k^* \leftarrow P_k$
  tant que $T > T_{min}$ **faire** % Boucle de décroissance de la température
    répète % Boucle de stabilisation
      si $T < T_{seuil}$ **alors**
        $P_k' \leftarrow modif1(P_k)$
      sinon
        $P_k' \leftarrow modif2(P_k)$
      fin si
      $\Delta e \leftarrow énergie(P_k') - énergie(P_k)$
      si $(\Delta e \leq 0)$ ou $(proba\_Boltzmann(\Delta e) >$ nombre aléatoire $x \in [0; 1])$ **alors**
        $P_k \leftarrow P_k'$
        si $énergie(P_k^*) > énergie(P_k)$ **alors**
          $P_k^* \leftarrow P_k$
        fin si
      sinon
        $nb\_états\_refusés \leftarrow 1 + nb\_états\_refusés$
      fin si
    jusqu'à ce que $nb\_états\_refusés < min\_états\_refusés$
    $T \leftarrow abaisser(T)$
  fin tant que
  retourner P_k^*
fin Procédure

La partition initiale nécessaire au recuit est faite soit de manière aléatoire, soit grâce à la méthode de percolation décrite section 2.7.1. Les fonctions $proba\_Boltzmann$ et $abaisser$ de l'algorithme 12 sont décrites dans la sous-section précédente (*supra* 3.2.1). La fonction $énergie$ est la fonction objectif à minimiser.

Les paramètres de cet algorithme sont les suivants :

– les températures maximale et minimale du recuit : T_{max} et T_{min} ;
– le facteur de la fonction de décroissance de la température : T_{fact} ;
– la température seuil qui permet de savoir si la température est basse ou élevée : T_{seuil} ;
– le nombre minimal d'états refusés avant de considérer que l'état est figé : $min\_états\_refusés$.

Seconde adaptation

Nous avons présenté cette seconde adaptation de la métaheuristique du recuit simulé au problème du partitionnement de graphe non contraint dans [BIC07]. Cette adaptation est basée sur l'article de David Johnson, Cecilia Aragon, Lyle McGeoch et Catherine Schevon, [JOH89]. Celui-ci présente une adaptation du recuit simulé à la bissection de graphe. Dans le but de respecter la contrainte sur la balance de partitionnement, la fonction de coût utilisée dans cet article est la suivante :

$$f(S_1, S_2) = coupe(S_1, S_2) + \alpha \left(poids(S_1) - poids(S_2)\right)^2 \ , \tag{3.1}$$

où S_1 et S_2 sont les deux parties de la bissection, $coupe$ est définie section 1.4, et α est une constante. C'est cette dernière qui est chargée de pénaliser la partition lorsque celle-ci ne respecte plus la balance de partitionnement.

Dans notre adaptation au partitionnement non contraint, nous avons remplacé dans la fonction objectif 3.1, le coût de coupe par le ratio de coupe. De plus, puisque nous cherchons une partition en k parties, $P_k = \{S_1, \ldots, S_k\}$, et non pas une bissection, la différence de poids entre les deux parties de l'équation 3.1 a été remplacée par la différence entre le poids maximal et le poids minimal des parties de la partition. Ainsi, nous avons utilisé la fonction de coût suivante :

$$ratio_\alpha(P_k) = ratio(P_k) + \alpha \left(\max_{i \in \{1, \ldots k\}} poids(S_i) - \min_{i \in \{1, \ldots k\}} poids(S_i) \right) \ . \tag{3.2}$$

L'algorithme 13 présente la seconde adaptation du recuit simulé au partitionnement non contraint inspirée par [JOH89]. Si la structure de cet algorithme reste la même que celle de l'algorithme 11, plusieurs détails changent :

– la boucle de décroissance de la température ne s'arrête pas quand la température a atteint une valeur seuil, mais lorsque $compteur\_arrêt = 5$. Ce compteur d'arrêt permet de simuler le refroidissement « extrême » du processus à partir duquel plus aucun nouveau minimum d'énergie ne peut être atteint ;
– le nombre d'itérations de la boucle de stabilisation est fixé à un nombre de grande taille : $mult\_taille * card(S)$ où $mult\_taille \in [2^{-2}; 2^{10}]$;
– une nouvelle partition est crée en déplaçant un sommet sélectionné aléatoirement vers une autre partie de la partition, elle aussi choisie aléatoirement ;
– le nombre de nouvelles partitions acceptées pendant une boucle de stabilisation est enregistré dans $nb\_mouvements$. Ce dernier permet avec $min\_pourcent$ d'évaluer si le système est froid, et dans ce cas d'incrémenter $compteur\_arrêt$. Compte tenu de la taille de $mult\_taille * card(S)$, $min\_pourcent$ sera pris dans $[10^{-4}; 10^{-1}]$.

Algorithme 13 Seconde adaptation du recuit simulé.

Procédure RECUIT2($G = (S, A)$, k, T_{max}, $mult\_taille$, $min\_pourcent$)
  $P_k \leftarrow$ partition aléatoire de G en k parties
  $T \leftarrow T_{max}$
  $compteur\_arrêt \leftarrow 0$
  tant que $compteur\_arrêt < 5$ **faire** % Boucle de décroissance de la température
    $nb\_mouvements \leftarrow 0$
    pour $i = 1$ à $mult\_taille * card(S)$ **faire** % Boucle de stabilisation
      $P'_k \leftarrow$ déplacement d'un sommet de P_k vers une autre partie
      $\Delta e \leftarrow ratio_\alpha(P'_k) - ratio_\alpha(P_k)$
      si ($\Delta e \leq 0$) ou ($proba\_Boltzmann(\Delta e) >$ nombre aléatoire $x \in [0; 1]$) **alors**
        $P_k \leftarrow P'_k$
        $compteur\_arrêt \leftarrow 0$
        $nb\_mouvements \leftarrow nb\_mouvements + 1$
      fin si
    fin pour
    $T \leftarrow abaisser(T)$
    si $nb\_mouvements < min\_pourcent * mult\_taille * card(S)$ **alors**
      $compteur\_arrêt \leftarrow compteur\_arrêt + 1$
    fin si
  fin tant que
  retourner P_k
fin Procédure

Dans [JOH89], le recuit simulé semble donner des résultats comparables à ceux de l'algorithme de Kernighan-Lin (*supra* 2.4). Cependant, son adaptation au partitionnement en k parties est problématique. Pour que le recuit fonctionne, il faut trouver un déplacement élémentaire efficace entre deux solutions admissibles. Le déplacement élémentaire consistant à déplacer un sommet entre deux parties présente plusieurs inconvénients. Après quelques itérations, la balance de partitionnement n'est plus respectée. Cependant, une fonction de coût trop restrictive sur la balance ne permet pas de trouver des partitions de bonne qualité. Si pour prévenir l'inconvénient précédent le choix des déplacements est restreint, cela bloque tellement le processus du recuit qu'il devient inefficace.

Les paramètres de cet algorithme sont les suivants :
- la température maximale du recuit : T_{max} ;
- le facteur de la fonction de décroissance de la température : T_{fact} ;
- le paramètre du nombre d'itérations de la boucle de stabilisation : $mult\_taille$;
- pour évaluer si le système est froid, on utilise $min\_pourcent$;
- le facteur permettant de faire respecter la contrainte de partitionnement : α.

3.3 Les algorithmes de colonies de fourmis

Dans cette section, nous introduirons d'abord brièvement le principe des algorithmes de colonies de fourmis, puis nous présenterons notre adaptation de ces algorithmes au problème du partitionnement de graphe non contraint.

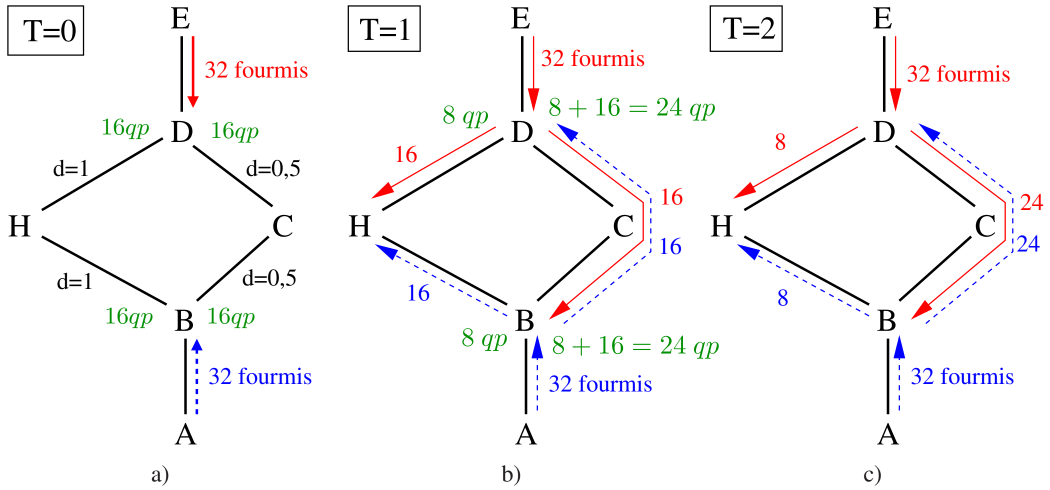


FIG. 3.1 – Un exemple de la stratégie des colonies de fourmis, le contournement d'obstacles. a) Le schéma initial avec les distances, 32 fourmis arrivent en D et en B. Pas de phéromone, les fourmis choisissent une direction de manière aléatoire. b) À t=1, évaporation de la phéromone, arrivée en B des fourmis venant de C. c) Les fourmis sont allées majoritairement dans la plus courte branche.

3.3.1 Introduction aux algorithmes de colonies de fourmis

L'idée de s'inspirer des colonies de fourmis pour résoudre un problème d'optimisation combinatoire est due à A. Colorni, M. Dorigo et V. Maniezzo [COL92]. L'optimisation par colonies de fourmis, [DOR99b, DOR99a, DOR02, DOR05] s'inspire du comportement des fourmis lorsque celles-ci sont à la recherche de nourriture. Une fourmi en se déplaçant pose de la phéromone, substance olfactive et volatile, sur son chemin. Les fourmis se dirigent de manière probabiliste en tenant compte de la quantité de phéromone qui est autour d'elles et qui a été précédemment déposée par les autres membres de la colonie. Plus la quantité de phéromone indiquant un chemin est grand, plus la fourmi a tendance à suivre ce chemin. Cependant, comme la phéromone s'évapore progressivement, le choix probabiliste que prend une fourmi pour choisir son chemin évolue continuellement. Dans la suite de cette section, nous désignerons par *qp* l'unité de mesure d'une quantité de phéromone « unitaire ».

Les algorithmes de colonies de fourmis s'inspirent de la capacité collective qu'ont les fourmis de résoudre certains problèmes, qu'elles ne pourraient pas résoudre individuellement étant donné leurs capacités limitées. Grâce à l'utilisation qu'elles font de leurs phéromones, les fourmis peuvent collectivement trouver le plus court chemin entre deux points. Ce mécanisme de choix entre chemins est illustré par l'exemple de la figure 3.1.

À T=0, 32 fourmis sont en D et 32 autres en B. Le choix du chemin à suivre entre H et C est complètement aléatoire puisqu'il n'y a encore aucune phéromone sur ce chemin. Chaque groupe de fourmis se divise donc en deux parties égales qui déposent chacune 16 *qp* sur leur chemin. À T=1, 32 nouvelles fourmis se présentent en D et 32 autres en B. Mais pendant ce temps, les 16 fourmis parties de D en passant par C sont arrivées en B, de même les 16 autres parties de B en passant par C sont en D. Il y a donc $16 + \frac{16}{2} = 24$ *qp* qui indiquent le chemin en passant par C. Ces $\frac{16}{2} = 8$ *qp* sont dues à l'évaporation. De l'autre côté, les 16 fourmis parties de D et les 16 autres parties de B se retrouvent en H. Il n'y a donc que $\frac{16}{2} = 8$ *qp* qui indiquent en D comme en B le chemin passant par H. Puisque le chemin passant par C est signalé par 26 *qp* et celui passant par D, par 8 *qp*, les nouvelles fourmis vont majoritairement choisir le chemin passant par C. C'est ce qui se passe à T=2. Dans cet exemple, quand la phéromone s'évapore, la quantité de phéromone est divisée par deux. Cependant, ce schéma de décroissance peut être différent.

Les colonies de fourmis sont basées sur deux concepts fondamentaux [DRÉ03] :

- la **stigmergie** : c'est la possibilité d'échanger des informations par le biais de modifications de l'environnement. C'est donc une forme de communication entre les individus d'un groupe. C'est entre autres par ce biais que les fourmis communiquent entre elles, notamment grâce aux phéromones ;
- l'**auto-organisation** de la population. Les fourmis prennent des décisions localement et ne sont pas soumises à des ordres hiérarchiques. L'auto-organisation de la population permet à celle-ci de trouver une solution vers laquelle elle n'a pas été orientée. Grâce à cette vision très locale de l'environnement, les algorithmes de colonies de fourmis sont très robustes (ils sont toujours performants, même en cas de défaillance de certains individus) et flexibles (une colonie peut s'adapter à un nouvel environnement).

Algorithme 14 Algorithme générique de l'optimisation par colonies de fourmis.

```
Procédure ColoniesDeFourmisGénérique
    répète % Boucle externe
        construction d'une solution par déplacement des fourmis
        évaporation de la phéromone
        affinage de la solution   % étape optionnelle
    jusqu'à ce que critère d'arrêt
    retourner P_k
fin Procédure
```

L'algorithme 14 présente de manière générique la méthode d'optimisation par colonies de fourmis [DOR02]. Les auteurs précisent que la planification et la synchronisation des trois étapes de la boucle externe sont à la discrétion de la personne qui implémentera leur méthode.

3.3.2 Notre adaptation des colonies de fourmis au partitionnement non contraint

Cette sous-section a pour objet de présenter l'adaptation de la méthode d'optimisation par colonies de fourmis au problème du partitionnement non contraint (*supra* 1.6). Nous avons choisi pour cette adaptation une approche peu classique, qui consiste à mettre en concurrence plusieurs colonies.

Soit un graphe $G = (S, A)$, ce graphe représente le territoire sur lequel peuvent évoluer les fourmis. Sur ce territoire est réparti de la nourriture. Les colonies sont en concurrence pour récupérer cette nourriture, ou plus exactement pour s'accaparer des régions de ce territoire possédant de la nourriture. Le but de chaque colonie est de créer une région riche en nourriture et dans laquelle les fourmis des autres colonies ne peuvent venir. Les correspondances entre le graphe et le territoire sur lequel évoluent les fourmis sont les suivantes :

- de nombreux champs composent le territoire. Chacun de ces champs possède une quantité de nourriture donnée ;
- à chaque arête du graphe correspond un champ, et la quantité de nourriture sur celui-ci est égale au poids de cette arête ;
- un sommet représente le carrefour entre plusieurs chemins menant aux champs ;
- les fourmis peuvent traverser les champs, mais s'arrêtent dans les carrefours ;
- les fourmis sont pacifistes, elles ne se combattent pas ;

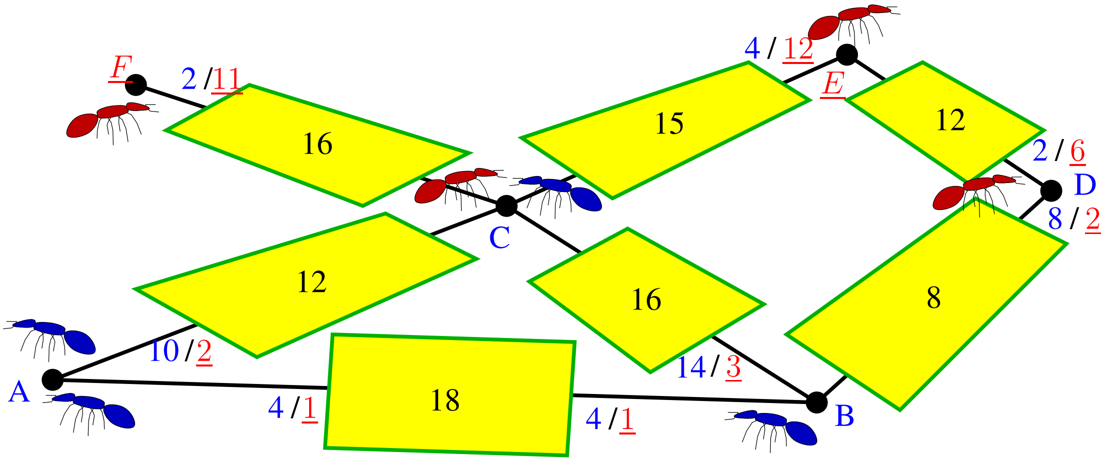


FIG. 3.2 – Illustration de l'adaptation des colonies de fourmis au partage de territoire sur un graphe.

- une fourmi choisit sa direction en tenant compte de la phéromone déposée sur les chemins entourant le carrefour sur lequel elle se trouve ;
- quand une fourmi se déplace, elle dépose la même quantité de phéromones sur les chemins qui sont à l'entrée et à la sortie du champ ;
- un carrefour appartient à une colonie lorsque la somme des quantité de phéromone signalant les chemins l'entourant dépasse celle de chacune des autres colonies ;
- si les deux carrefours menant à un champ appartiennent à une même colonie, alors on considère que celle-ci possède le champ ;
- la région possédée par une colonie est constituée de l'ensemble des carrefours appartenant à cette colonie.

La figure 3.2 illustre cette approche. Sur celle-ci, des fourmis de deux colonies, la colonie « rouge » (fourmis tournées vers la droite) et la colonie « bleue » (fourmis tournées vers la gauches), sont en compétition. Le graphe, formé des sommet de A à F, est recouvert de champs sur ses arêtes. Au centre de chacun des champs est notée sa réserve de nourriture. La quantité de phéromone déposée par les fourmis de chaque colonie est inscrite sur les chemins menant aux champs, dans les « couleurs » respectives des colonies (à la différence de la colonie bleu, la quantité de phéromone de la colonie rouge est souligné d'un trait). Ainsi, le chemin AC possède 10 qp bleues et 2 qp rouges, à l'entrée comme à la sortie du champ. Les noms des sommets de la colonie rouge sont souligné, pas ceux de la colonie bleu. L'exemple du sommet D rappelle qu'une fourmi peut être sur un sommet que sa colonie ne possède pas. En effet, le sommet D appartient à la colonie bleue car la quantité de phéromone du carrefour vaut 10 pour la colonie bleue, mais 8 pour la colonie rouge. De même, l'exemple du sommet C nous montre que des fourmis de plusieurs colonies différentes peuvent être au même carrefour. L'évaporation progressive de la phéromone permet à certains chemins de posséder de la phéromone d'une colonie qui ne possède aucun des carrefours auquel il aboutit.

Les régions possédées par les différentes colonies forment naturellement une partition du graphe. En effet, puisque chaque sommet du graphe appartient à une colonie, le graphe est partitionné en autant de parties qu'il y a de colonies. Ainsi, pour obtenir une k-partition du graphe, il suffit d'utiliser k colonies. La coupe de la partition sera constituée des champs bordés par des carrefours de colonies différentes.

Avant de choisir un chemin, les fourmis tiennent compte de la quantité de phéromone déposé sur celui-ci, en utilisant le même mécanisme que celui de l'exemple de la figure 3.1, à un détail près.

En effet, dans le cas où une fourmi arrive à un carrefour possédant un chemin sans phéromone de sa colonie, elle n'a aucune chance de l'emprunter. Pour pallier à cet inconvénient, une quantité de phéromone fictive est ajouté pour chaque chemin, lors du calcul des probabilités.

Algorithme 15 Colonies de fourmis en compétition pour le partitionnement de graphe non contraint.

Procédure COLONIESDEFOURMIS($G = (S, A)$, k)
  $P_k \leftarrow$ initialisation de la partition en k parties
  répète % Boucle externe
    pour tout colonie de fourmis c_i **faire**
      pour tout fourmi f_j^i de la colonie c_i **faire**
        choisir un chemin partant du carrefour où se trouve la fourmi f_j^i
        f_j^i dépose une quantité de phéromone égale à celle de nourriture du champ
        si au moins un sommet change de partie à cause de la quantité de phéromone **alors**
          $P'_k \leftarrow$ mettre à jour la partition en fonction des sommets déplacés
          si $f(P'_k) < f(P_k)$ ou $h(P'_k|P_k) > random(1)$ **alors**
            conserver le nouveau déplacement
            $P_k \leftarrow P'_k$
          sinon
            rejeter le déplacement et considérer que la fourmi f_j^i est restée immobile
          fin si
        fin si
      fin pour
    fin pour
    évaporation de la phéromone
  jusqu'à ce que critère d'arrêt
  retourner P_k
fin Procédure

L'algorithme 15 présente le fonctionnement de cette adaptation des colonies de fourmis au partitionnement non contraint. La première étape de celui-ci consiste à créer une partition initiale en k parties du graphe. C'est à partir de cette partition que les fourmis seront disposées par colonies sur le graphe. Comme une colonie est très peu efficace lorsque les fourmis qui la constituent sont dispersées, il est préférable que cette première partition crée des parties connexes. Pour cela, nous utilisons l'algorithme de percolation présenté en section 2.7.1.

Une fois la partition P_k initialisée avec le résultat trouvé par la méthode de percolation, l'algorithme itère autour d'une boucle externe dont le critère d'arrêt est le nombre d'itérations. Cette boucle externe a pour but de déplacer les colonies les unes après les autres, et dans chaque colonie, fourmi après fourmi. Ainsi, dans la boucle externe deux boucles sont imbriquées, la première itère sur chaque colonie, la seconde à l'intérieur de la première itère sur chaque fourmi. Le déplacement d'une fourmi se fait en plusieurs étapes :

- **choix d'un chemin**. Avant de se déplacer, une fourmi est à un carrefour. La première étape de ce déplacement consiste donc à choisir un des chemins partant de ce carrefour. Comme nous l'avons vu, ce choix se fait de manière probabiliste en tenant compte de la quantité de phéromone posée sur ces chemins, mais aussi d'une quantité de phéromone fictive nb_f pour n'écarter *a priori* aucun chemin ;
- **traversée du champ**. Une fois le chemin choisi, la fourmi l'emprunte pour arriver sur le champ. La nature du champ lui donne l'indication de la nourriture qui est dessus, c'est-à-dire le poids

de l'arête sous-jacente. La fourmi dépose alors à l'entrée et à la sortie du champ une quantité de phéromone égale à cette quantité de nourriture. Puis la fourmi s'arrête au carrefour situé à la sortie du champ ;
- **déplacement de sommets**. Une nouvelle quantité de phéromone ayant été posées, l'appartenance des carrefours, *i.e.* des sommets, d'origine et de destination de la fourmi peut avoir changé. Deux situations se présentent alors :
  - aucun déplacement de sommet n'a eu lieu. Le déplacement de la fourmi est alors terminé, P_k est inchangé et le déplacement d'une nouvelle fourmi peut commencer ;
  - sinon, le changement de partie d'un ou des deux sommets est répercuté dans P'_k, partition identique à P_k sauf en ce qui concerne ces deux sommets. Le coût des partitions P'_k et P_k est calculé puis comparé, et dans le cas où le coût de P'_k est inférieur à celui de P_k, ou dans le cas où la partition P'_k est acceptée avec une probabilité de Boltzmann, alors le déplacement de la fourmi est acceptée et P_k reçoit P'_k. Dans le cas contraire, le déplacement est rejeté et la fourmi est renvoyée à son carrefour d'origine en supprimant la quantité de phéromone déposée.

La fonction de probabilité de Boltzmann utilisée est la suivante :

$$h(P'_k|P_k) = \exp^{-\frac{f(P'_k)-f(P_k)}{T}} ,$$

où T est une constante qui permet d'ajuster cette fonction.

Une fois les déplacements des fourmis des différentes colonies effectués, la phéromone s'évapore. C'est la dernière étape de la boucle externe. La quantité de phéromone qui s'évapore est fonction du temps depuis laquelle elle a été posée, mais aussi d'un paramètre t_{max} qui limite la « durée de vie » d'une phéromone. Nous devons faire remarquer que le choix d'un chemin par une fourmi ne tient pas compte de la quantité de phéromone placées au même « tour » (même itération de la boucle externe), mais seulement de la quantité de phéromone placée aux tours précédents. Cependant, le déplacement d'un sommet entre deux parties tient compte de toutes les quantités de phéromone présentent, y compris celles placées précédemment pendant le même tour. La quantité de phéromone posée sur un chemin se calcul ainsi :

$$\textit{phéromones}(A \rightarrow B) = \sum_{t=1}^{t_{max}} \frac{1}{t} * pose(t) ,$$

où $pose(t)$ est la quantité de phéromones déposées il y a t tours.

Les variables qui sont utiles à notre implémentation de l'optimisation par colonies de fourmis sont les suivantes :
- le nombre k de colonies ;
- le nombre de fourmis par colonies ;
- t_{max} qui fixe la « durée de vie » d'une phéromone ;
- une constante T qui permet d'ajuster la fonction de probabilité de Boltzmann ;
- une quantité de phéromones fictives nb_f qui permet de n'exclure aucun chemin lors du choix d'une destination.

Pour conclure sur les colonies de fourmis, nous pouvons dire que notre adaptation est une méthode hybride entre colonies de fourmis et recuit simulé, puisque nous utilisons la règle d'acceptation de Metropolis (fonction de probabilité de Boltzmann) entre chaque déplacement de fourmis.

3.4 Les algorithmes évolutionnaires

Les algorithmes évolutionnaires forment une famille de métaheuristiques « distribuées », c'est-à-dire basées sur une population [DRÉ03]. Ces algorithmes ont comme point commun de s'inspirer du principe de sélection naturelle de Charles Darwin. La théorie de la sélection naturelle s'applique aux individus d'une population et repose sur trois principes :

- **la variation** : les individus diffèrent les uns des autres ;
- **l'adaptation** : les individus les mieux adaptés à leur environnement vivent plus longtemps et procréent plus ;
- **l'hérédité** : les caractéristiques des individus sont héréditaires.

Le principe de la sélection naturelle est le suivant : puisque tous les individus sont différents et que certains, mieux adaptés, vont plus procréer, les caractéristiques avantageuses de ces derniers seront héritées par les générations suivantes et, avec le temps, deviendront dominantes dans la population.

Les algorithmes évolutionnaires sont apparus dès la fin des années 1950 [FRA57, HOL62, FOG66]. À l'origine, trois approches des algorithmes évolutionnaires ont émergé indépendamment. Les algorithmes génétiques sont certainement l'approche la plus connue, notamment grâce à l'ouvrage de référence de David Goldberg [GOL89]. Les algorithmes génétiques respectent le shéma d'un algorithme évbolutionnaire avec une particularité : lors des étapes d'évaluation, une solution du problème est entièrement construite à partir du génotype d'un unique individu (celui-ci est souvent une chaine de symboles binaires). Cependant, d'autres branches existent comme la programmation évolutionnaire, qui s'intéresse aux liens comportementaux entre des objets parents et leurs héritiers, en utilisant les mécanismes de sélection et de mutation, ou comme les stratégies d'évolution qui sont une méthode d'optimisation de paramètres variant continûment. Les chercheurs de ces différentes « écoles » ont combiné ces différentes approches dans une branche unique, les algorithmes évolutionnaires.

Dans une première sous-section nous étudierons les algorithmes évolutionnaires, puis, dans une seconde sous-section, nous introduirons les adaptations de ces algorithmes que nous avons faites pour les problèmes du partitionnement de graphe contraint et non contraint.

3.4.1 Principe des algorithmes évolutionnaires

Comme dans le cas des autres métaheuristiques, les algorithmes évolutionnaires sont utilisés pour résoudre des problèmes d'optimisation combinatoire (*supra* 1.1). De par leur origine, tout un vocabulaire issu de la biologie est utilisé pour décrire ces méthodes. Considérant un problème d'optimisation combinatoire, une population est un ensemble de points de l'espace des solutions, chacun de ces points est appelé individu. Un individu est constitué d'un patrimoine génétique qui le caractérise et le différencie des autres individus ; concrètement les gènes sont les blocs élémentaires caractérisant une solution. La fonction objectif évalue l'adaptation d'un individu à son environnement et est donc appelée fonction d'adaptation ou *fitness*. Les algorithmes évolutionnaires ont pour but de maximiser la fonction d'adaptation. Dans le cas d'une fonction objectif à minimiser, en supposant qu'elle est positive, l'inverse de cette fonction peut être utilisée comme fonction d'adaptation.

Algorithme

L'algorithme 16 présente de manière générique les étapes d'un algorithme évolutionnaire. Dans un tel algorithme, la population évolue de manière itérative. Les grandes étapes de cette évolution sont les suivantes :

1. une population initiale de $nb\_individus$ est créée. De cette population dériveront toutes les futures générations. Il est donc préférable qu'elle couvre une large partie de l'espace des solutions du problème ; c'est pourquoi elle est souvent créée de manière aléatoire ;

2. une boucle itérative permet de faire évoluer la population en générations successives avec l'aide de trois principes :
   – l'**adaptation** d'un individu qui correspond à la valeur relative de la fonction objectif entre cet individu et les autres ;
   – la **sélection** des individus de la population en deux étapes. Premièrement, la sélection des futurs parents, c'est-à-dire ceux qui vont se reproduire. Ce qui ne veut pas dire que les autres membres de la population vont disparaître car c'est l'objet de la seconde sélection. Cette première étape est assurée par l'opérateur de sélection. Secondement, la sélection de la population qui sera présente à la prochaine itération. Cette sélection est assurée par l'opérateur de remplacement ;
   – la **diversification** de la population. C'est ce qui va permettre d'explorer l'espace des solutions. Les parents sélectionnés ont deux possibilités pour engendrer des enfants : le croisement ou la mutation. Les individus créés seront les enfants de cette itération.

3. un critère d'arrêt qui met fin à la recherche. Il peut être fonction du nombre de générations, de l'adaptation du meilleur individu ou encore du temps passé.

Algorithme 16 Algorithme évolutionnaire.

Procédure ALGOÉVO($population\_initiale$, $nb\_croisements$, $nb\_mutations$)
  $population \leftarrow population\_initiale$
  $nb\_individus \leftarrow card(population)$
  évaluation de l'adaptation des $nb\_individus$ éléments de $population$
  $nb\_enfants \leftarrow nb\_croisements + nb\_mutations$
  répète
    $parents \leftarrow$ sélection d'individus de $population$ pour la reproduction
    $enfants \leftarrow \emptyset$
    pour $i = 1$ à $nb\_croisements$ **faire**
      $enfant \leftarrow$ croisement de plusieurs individus de $parents$
      évaluation de l'adaptation de $enfant$
      $enfants \leftarrow \{enfant\} \cup enfants$
    fin pour
    pour $i = 1$ à $nb\_mutations$ **faire**
      $enfant \leftarrow$ mutation d'un individu de $parents$
      évaluation de l'adaptation de $enfant$
      $enfants \leftarrow \{enfant\} \cup enfants$
    fin pour
    $population \leftarrow$ remplacement par $nb\_individus$ venant de $enfants$ et $parents$
  jusqu'à ce que Critère d'arrêt
  retourner individu le mieux adapté de $population$
fin Procédure

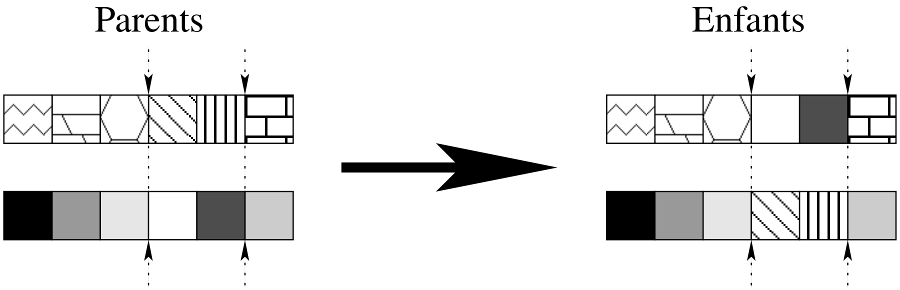


FIG. 3.3 – Illustration du travail d'un opérateur de croisement. Les gènes sont représentés par des carrés.

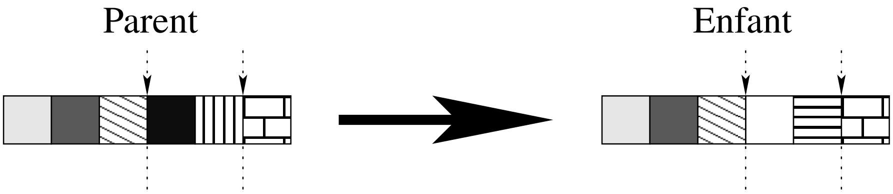


FIG. 3.4 – Illustration du travail d'un opérateur de mutation. Les gènes sont représentés par des carrés.

Opérateurs de diversification

La diversification va permettre à l'algorithme d'explorer l'espace des solutions à la recherche du ou des optima globaux. Cette étape consiste à créer de nouvelles solutions (les enfants), à partir des solutions présentes au début de l'itération et sélectionnées à l'étape précédente (les parents). Les algorithmes évolutionnaires se sont inspirés des méthodes de reproduction du vivant pour créer des opérateurs de diversification. Le croisement joue le rôle de la reproduction sexuée et la mutation celui de la reproduction asexuée :

- le **croisement** nécessite plusieurs parents, en général deux, pour produire plusieurs enfants, en général autant que de parents. Ce opérateur combine les gènes (blocs élémentaires caractérisant une solution) des parents pour créer les enfants. L'idée sous-jacente est qu'il va être possible de remplacer quelques gènes « pauvres » d'un parent par les gènes correspondants plus « riches » d'un autre parent. Cet opérateur n'est donc pas nécessairement aléatoire. Une illustration d'un opérateur standard de croisement est présentée figure 3.4.1 ;
- la **mutation** consiste à modifier localement un ou plusieurs éléments de la population. C'est un opérateur très simple mais essentiel au fonctionnement des algorithmes évolutionnaires, car il assure la diversification de la population. Dans certaines implémentations, seul cet opérateur est utilisé [FOG66]. Une façon simple de créer une mutation est de modifier aléatoirement un ou plusieurs gènes d'un individu en respectant les contraintes du problème. Une illustration d'un opérateur standard de mutation est présentée figure 3.4.1.

L'opérateur de mutation est souvent plus utilisé pour explorer l'espace des solutions et l'opérateur de croisement pour atteindre les optima locaux. Ainsi, lorsque la mutation est suffisamment aléatoire, les croisements peuvent être plus déterministes. Les opérateurs de croisement et de mutation sont délicats à régler. Trop simple et/ou pas assez nombreuse la mutation va s'enfermer dans un optimum local ; trop compliquée et/ou trop nombreuse elle va empêcher la convergence de l'algorithme. De plus, en fonction de la convergence de l'algorithme, une diversification adaptée peut être utile. Dans le but de minimiser ces différents problèmes, des opérateurs adaptatifs peuvent être utilisés [EIB99, YAN02].

Sélection des individus

D'une manière générale, la capacité d'un individu à être sélectionné, que ce soit pour la reproduction ou le remplacement, dépend de son adaptation. L'opérateur de sélection comme de remplacement est chargé de déterminer le nombre de sélections de chaque individu en fonction de son adaptation.

Les individus les mieux adaptés se reproduisent plus souvent et remplacent les moins bons. Ainsi, les copies du meilleur individu vont envahir la population à partir d'un certain temps appelé temps de domination. Plus ce temps sera court, plus la pression sélective sera élevée. Dans un tel cas, il y a un fort risque de convergence prématurée vers un optimum local. En effet, un super-individu envahira la population et l'exploration de l'espace de recherche deviendra alors locale, centrée sur lui. Dans le cas contraire, *i.e.* pression sélective trop faible, l'algorithme ne convergera pas. Les individus se répandront dans l'espace des solutions sans progresser.

La pression sélective, très importante dans le cas des algorithmes évolutionnaires est principalement déterminée par les opérateurs de sélection et de remplacement.

Opérateurs de sélection

L'opérateur de sélection a pour mission de choisir dans la population présente les futurs parents nécessaires à l'étape de remplacement. Plusieurs opérateurs de sélection existent :

- **sélection par tournoi** : une loi de probabilité uniforme est utilisée pour sélectionner des couples d'individus dans la population. Dans chaque couple sélectionné, les deux individus « combattent » l'un contre l'autre. Lors de chaque combat, l'individu le mieux adapté l'emporte avec une probabilité dans $]0,5;1]$ qui est un paramètre de la méthode permettant le réglage de la pression sélective. Dans ce type de sélection, un super individu sera sélectionné plusieurs fois et prendra la place de plusieurs parents ;
- **sélection par roulette** : c'est une méthode proportionnelle. Son nom vient de l'analogie avec le tirage aléatoire des roulettes de casino. À chaque individu i est associée une probabilité d'être choisie proportionnelle à son adaptation $f(i)$: $p_i = \frac{f(i)}{\sum_j f(j)}$. Puis $nb\_parents$ tirages aléatoires dans $[0;1]$ permettent de sélectionner les futurs parents. La variance de ce processus est élevée. Dans le pire cas, il est possible qu'aucun individu bien adapté ne soit sélectionné. Ce phénomène, appelé biais de sélection, peut être limité avec une population nombreuse. Une variante appelée sélection par échantillonnage stochastique universel [DRÉ03] permet elle aussi de réduire ce biais ;
- **reste stochastique sans remplacement** : elle associe sélection par roulette et sélection déterministe. Un nombre minimal de représentants de chaque individu parmi les futurs parents est déterminé par avance en fonction de l'adaptation de l'individu, puis la population est complétée par tirages aléatoires :
  - le nombre minimal de représentants d'un individu i est donné par la partie entière : $\lfloor \frac{f(i)}{\sum_j f(j)} \rfloor$;
  - le reste des parents est complété grâce à la sélection par roulette où la valeur d'adaptation de chaque individu est remplacée par : $f(i) \leftarrow f(i) - \lfloor \frac{f(i)}{\sum_j f(j)} \rfloor$.

Opérateurs de remplacement

L'opérateur de remplacement a pour rôle d'assurer la prolifération des meilleurs individus au détriment des plus mauvais. Au cours de cette étape, la probabilité de survie de chaque individu doit

donc correspondre à son adaptation. Plusieurs opérateurs de remplacement existent :
- le **remplacement par génération** consiste à ne conserver que les enfants dans la génération suivante. Dans ce cas, $nb\_enfants = nb\_individus$. Les algorithmes génétiques d'origine utilisent ce remplacement ;
- le **remplacement des stratégies d'évolution** est une sélection déterministe des $nb\_individus$ meilleurs individus parmi l'ensemble des enfants ou parmi l'ensemble des parents et des enfants. Dans ce cas, le nombre d'enfants est plus grand que celui d'individus de la population ;
- l'**élitisme** consiste à conserver au moins un individu possédant la meilleure adaptation. Cet opérateur est couplé avec un autre opérateur qui sélectionne le reste de la population. L'élitisme peut améliorer grandement les performances d'un algorithme évolutionnaire, comme être très néfaste [DEJ93] car induisant un pression sélective trop grande.

Il faut distinguer l'opérateur de sélection de celui de remplacement, l'opérateur de sélection est chargé de sélectionner les parents parmi la population, alors que l'opérateur de remplacement détermine la nouvelle population à partir des enfants et des parents

Paramétrage

Les algorithmes évolutionnaires répondent à nombre important de paramètres. Du réglage de ceux-ci, découlent la rapidité et la qualité de la convergence. Ces paramètres sont souvent difficiles à régler. Ils sont énumérés ci-après :
- la taille de la population, $nb\_individus$, est un paramètre important de l'optimisation. Plus elle sera grande, plus la probabilité de convergence de l'algorithme sera élevée. Cependant, plus il y aura d'individus, plus l'algorithme sera lent car la complexité de l'algorithme est directement proportionnelle à cette taille ;
- lors de l'étape de diversification, $nb\_mutations$ individus subissent une mutation ;
- $nb\_croisements$ détermine le nombre de croisements effectués à chaque itération ;
- le ou les paramètres liés au critère d'arrêt.
- d'autres paramètres potentiels, comme la probabilité de la sélection par tournoi ou encore pour l'élitisme le nombre d'individus conservé entre les générations.

Convergence et améliorations classiques

Des études sur la convergence des algorithmes évolutionnaires ont été menées. Les premières se sont limitées au cas du codage binaire des gènes des individus [GOL89] et permettent de mieux comprendre les effets de certains opérateurs de sélection, de croisement et de mutation. D'autres études ont été réalisées dans le cas du codage réel [CER94]. Elles permettent de montrer que la convergence peut être obtenue sans l'opérateur de croisement (ce qui revient à une sorte de recuit simulé en parallèle), et aussi d'établir le comportement asymptotique des ces algorithmes.

Plusieurs méthodes maintenant classiques permettent d'améliorer la rapidité et l'efficacité des algorithmes évolutionnaires :
- la **mise à l'échelle**, ou *scaling* en anglais : elle s'applique à la fonction d'adaptation dans le cas où les techniques de sélection peuvent être biaisées par les solutions trouvées. La figure 3.5 illustre en se plaçant dans le cas continu les deux problèmes qui peuvent se poser. Dans la figure de gauche, l'individu i possède une valeur d'adaptation très élevée comparée aux autres individus et l'algorithme risque de converger vers cet optimum local. Dans la figure de droite, aucun optimum ne se détâche et il serait intéressant de « grossir » artificiellement la zone entourée. La mise à l'échelle joue sur la modification de la valeur d'adaptation des individus ;

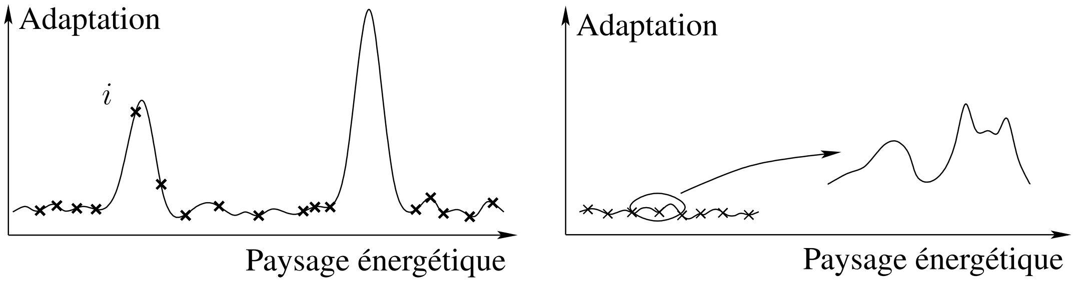


FIG. 3.5 – Exemples pour lesquels la mise à l'échelle peut s'avérer utile.

– **répartition**, ou *sharing* en anglais : elle a pour objectif de forcer la population à se répartir dans différentes régions de l'espace des solutions grâce à une notion de distance. Son utilisation est particulièrement recommandée dans le cas des fonctions objectifs possédant de nombreux optima locaux. Une amélioration importante de cette méthode est la répartition par région ou par *clusters* qui répartit les individus dans des régions de proximité ;
– l'**optimisation locale** : les algorithmes évolutionnaires permettent de trouver dans des espaces de grande dimension des solutions faisant partie d'un ou de plusieurs puits énergétiques (*supra* 1.1). Cependant ces solutions ne sont pas forcément le ou les minima globaux de ces puits. Atteindre ces minima nécessite l'utilisation d'algorithmes d'optimisation locale. Un algorithme d'optimisation local peut être utilisé une fois l'algorithme évolutionnaire fini ou pendant l'étape de diversification avec un recuit-simulé par exemple.

Pour plus de détails sur ces méthodes, se référer à [DUR04, GOT04].

3.4.2 Nos adaptations des algorithmes évolutionnaires au partitionnement de graphe

Il existe déja et depuis longtemps, plusieurs adaptations des algorithmes évolutionnaires au partitionnement de graphe [TAL91, ALP96c, MAI94]. Le partitionnement de graphe est difficilement parallélisable, cependant, le parallélisme structurel de la programmation évolutionnaire permet actuellement de créer avec succès des partitions avec un algorithme multi-niveaux parallèle [CHE06].

Plusieurs adaptations des algorithmes évolutionnaires au partitionnement de graphe ont été réalisées au cours de cette thèse. Cependant toutes ces adaptations ont donné des résultats peu performants en comparaison de ceux obtenus avec les autres méthodes comme le multi-niveaux (*supra* 2.6), le recuit simulé (*supra* 3.2) ou les algorithmes de colonies de fourmis (*supra* 3.3).

Adaptation générique d'un algorithme évolutionnaire au partitionnement de graphe

Ce paragraphe présente une approche générique de l'adaptation d'un algorithme évolutionnaire au partitionnement de graphe. Dans le cas du partitionnement de graphe, les individus d'un algorithme évolutionnaire sont des partitions du graphe et la population une famille de partitions.

Comme nous l'avons vu dans la sous-section précédente (*supra* 3.4.1), et au vu de l'algorithme 16, les algorithmes évolutionnaires utilisent 5 opérateurs essentiels suivants :

– la fonction d'adaptation ;
– l'opérateur de sélection ;
– l'opérateur de remplacement ;
– l'opérateur de croisement ;
– l'opérateur de mutation.

Dans le cadre de l'adaptation au partitionnement de graphe, on peut considérer que ces fonctions peuvent être facielement réalisées, mis à part l'opérateur de croisement. En effet, un opérateur de croisement nécessite souvent de pouvoir comparer deux gènes, or il est difficile de comparer 2 partitions, et plus précisément de comparer 2 parties de partitions différentes. Ce problème est amplifié quand k augmente.

Le problème du partitionnement de graphe, qu'il soit contraint ou non, est un problème de minimisation d'une fonction objectif (*supra* 1). Or, les différentes fonctions objectifs pour le partitionnement de graphe sont toutes positives si elles sont appliquées à une partition d'un graphe connexe. Dans le cas des graphes non connexes, on peut supposer que les partitions triviales de ces graphes, constituées des différentes parties connexes, ou regroupements de telles parties ne sont pas incluses dans les solutions admissibles du problème (commencer par une vérification, si besoin est). Ainsi, puisque les différentes fonctions objectifs sont positives, leurs inverses le sont aussi et le problème de minimisation devient un problème de maximisation. La fonction d'adaptation peut alors être égale à l'inverse de la fonction objectif.

En considérant l'inverse de la fonction objectif comme fonction d'adaptation, les différents opérateurs de sélection comme de remplacement présentés plus haut peuvent être utilisés. L'opérateur de sélection utilisé est le reste stochastique sans remplacement. L'opérateur de remplacement utilise l'élitisme et le remplacement des stratégies d'évolution. Le critère d'arrêt est *a priori* indépendant du problème traité, à savoir le partitionnement, et est choisi comme étant un temps minimum d'exécution.

Les paragraphes suivants présentent nos différentes adaptations des opérateurs de croisement et de mutation, au partitionnement de graphe, puisque les opérateurs de sélection, de remplacement et la fonction d'adaptation ont déjà été choisis.

Première adaptation au partitionnement de graphe non contraint

Dans ce paragraphe, nous présentons la première adaptation des algorithmes évolutionnaires au partitionnement de graphe non contraint.

L'opérateur de croisement de cette adaptation est basé sur l'échange de parties entre différentes partitions. En effet, une partition se définit entièrement par ses parties. Les parties d'une partition peuvent donc constituer les gènes d'une partition. La difficulté rencontrée est d'obtenir un croisement performant des ces gènes.

Cet opérateur de croisement a initialement été réalisé par Jean-Marc Alliot (directeur du Laboratoire d'optimisation globale dans lequel c'est passé cette thèse) et est implémenté dans l'algorithme évolutionnaire de [RUL03]. Dans cette approche, une partie est définie de manière déterministe par un algorithme d'expansion de région utilisant un mécanisme d'agrégation par voisinage (de type expansion de région, *supra* 2.2). Cet algorithme permet de créer une k-partition à partir d'un ensemble de k sommets du graphe. Chaque individu de la population va être caractérisé par l'ensemble des k sommets permettant de définir la partition correspondante, c'est-à-dire qu'un gène se définit par un sommet.

Cet opérateur de croisement engendre deux enfants à partir de deux parents. Les enfants sont constitués à partir du parcours simultané des gènes des parents. Pour autant, les gènes ne sont pas ordonnés avant ce parcours, *i.e.* lorsque deux gènes sont sélectionnés, un pour chaque parent, les parties correspondantes peuvent ne contenir aucun sommet en commun. Lors du parcours des gènes permettant de constituer le premier enfant, est retenu de chaque couple de gènes, le gène dont la partie correspondante possède le coût de coupe le moins élevé (*supra* 1.4). Pour le second enfant, est retenu un gène au hasard parmi chaque couple de gènes.

L'opérateur de mutation consiste à remplacer un des gènes s choisi de manière aléatoire, par un sommet sélectionné aléatoirement dans l'ensemble des sommets $S-\{s\}$. Le sommet s sélectionné devient le sommet initial d'où part la percolation qui définit cette partie. Après que s ait été choisi, k percolations sont effectuées à partir des sommets-gènes, pour définir la partition retournée par l'opérateur de mutation.

Cependant, le problème de cette première approche est qu'elle ne permet pas d'explorer la totalité de l'espace des solutions. En effet, elle utilise une méthode d'expansion de région déterministe pour constituer les parties d'une partition.

Seconde adaptation au partitionnement de graphe non contraint

Dans cette seconde adaptation, nous avons utilisé un opérateur de croisement très classique. Il est entre autres présenté dans [TAL91, MAI94, MAN96, ALA99, APE02]. Dans cette adaptation, un individu, c'est-à-dire une partition, est défini par des gènes qui représentent l'appartenance d'un sommet du graphe à une partie de la partition. Cet opérateur est une application directe de l'opérateur de croisement illustré par la figure 3.4.1. Deux enfants sont créés à partir de deux parents par échange de deux ensembles de gènes. Deux sommets s_1 et s_2 sont sélectionnés aléatoirement dans le graphe, le premier enfant hérite des gènes du second parent compris entre les sommets s_1 et s_2 et du premier parent pour le reste. Pour le second enfant, c'est l'inverse, il hérite des gènes du premier parent entre les sommets s_1 et s_2.

L'opérateur de mutation peut être choisi simplement comme dans le cas du recuit simulé (*supra* 3.2). En effet, une variation élémentaire d'une partition consiste à déplacer un sommet d'une partie de la partition vers une autre. Cela revient à considérer que pour un individu, c'est-à-dire une partition, chaque gène représente l'appartenance d'un sommet du graphe à une partie de la partition. Un opérateur de mutation basé sur ce principe est très facile à mettre en œuvre. Une mutation peut consister à déplacer une quantité fixe de sommets entre les parties. Chacun des sommets qui va être déplacé est choisi aléatoirement, et sa partie de destination est tirée au hasard.

Un opérateur de mutation adaptatif peut aussi être mis en place. Un tel opérateur permettrait de diminuer progressivement le nombre de déplacements de sommets de l'opérateur de mutation que nous venons de décrire, en fonction de l'avancement de l'algorithme.

Nous avons implémenté un algorithme évolutionnaire avec ce second opérateur de croisement et cet opérateur de mutation. Puis nous avons appliqué cet algorithme au problème du découpage de l'espace aérien européen (*infra* 5). Cependant, les résultats trouvés par cet algorithme étant significativement moins bons que ceux trouvés avec la version de Jean-Marc Alliot, nous avons rapidement cherché une autre approche pour l'adaptation des algorithmes évolutionnaires au partitionnement de graphe. Nos efforts se sont alors concentrés sur le partitionnement de graphe contraint, qui est l'objet de la plupart des publications présentant des méthodes évolutionnaires pour le partitionnement de graphe. Après les mauvais résultats trouvés par cette seconde adaptation, nous voulions pouvoir comparer plus facilement notre approche avec ces travaux, afin de valider notre travail.

Adaptation au partitionnement contraint

Dans notre adaptation des algorithmes évolutionnaires au partitionnement de graphe contraint, nous avons choisi comme structure génétique que chaque gène d'un individu (un individu est toujours une partition) représenterait l'appartenance d'un sommet du graphe à une partie de la partition. C'est la structure que nous avions utilisée pour décrire l'opérateur de mutation de notre première adaptation. Comme nous l'avions fait remarquer, si avec une telle représentation du génome, un opérateur de mutation est facile à définir, un opérateur de croisement est plus compliqué à mettre en place.

En effet, pour définir un opérateur de croisement entre deux partitions P_1 et P_2, il faut connaître les similitudes de découpage entre ces deux partitions. De manière plus précise, il faut reconnaître les liens de correspondance entre les parties des deux partitions, c'est-à-dire hiérarchiser les parties d'une partition par degré de similitudes avec les parties de l'autre partition. Nous avons créé un algorithme qui permet de hiérarchiser le degré de similitudes des parties de deux partitions entre elles. On appelle degré de similitude, le nombre de sommets appartenant aux deux parties des deux partitions. Il suffit alors de classer puis de sélectionner les couples de parties par degrés de similitude décroissants, pour obtenir une partition commune des deux partitions. Cette partition commune ne couvre cependant pas tous les graphes, il existe donc un ensemble de sommets appartenant à des parties différentes dans les deux partitions. C'est à partir de cet ensemble de sommets que va s'effectuer l'opération de croisement.

Le principe de l'opération de croisement est simple : créer un enfant dont la partition sera basée sur la partition commune aux deux parents. Tous les sommets de la liste des sommets appartenant à des parties différentes des deux partitions parentes sont affectés aux différentes parties de la partition enfant. L'affectation consiste à insérer chaque sommet dans l'une des deux parties choisie aléatoirement auquel il appartenait dans les partitions parentes. Cependant, dans le but de respecter la balance de partitionnement, quand au moins l'une des deux parties en question est pleine dans la partition enfant, une partie choisie au hasard parmi les parties non encore remplies est sélectionnée.

Dans le cas des graphes de grandes tailles, un mécanisme multi-niveaux (*supra* 2.6) est utilisé pour rendre le croisement plus rapide. L'algorithme de contraction utilisé pour regrouper les sommets entre eux est l'algorithme *Heavy Edge Matching* [KAR98b] (*supra* 2.6.3). Une fois les sommets agrégés, leur partie d'appartenance correspond à la partie à laquelle le plus grand nombre de sommets appartient. Une fois que l'appartenance de chaque nouveau sommet est établie, le degré de similitude entre chaque partie est calculé, puis l'opérateur de croisement est utilisé pour engendrer une nouvelle partition du graphe des sommets agrégés. Cette partition est alors directement reprojetée sur le graphe d'origine. Afin de gagner de la rapidité, aucune opération d'affinage lors de la projection sur le graphe d'origine n'est réalisée. Cependant ce choix que nous avions fait nous paraît plus discutable aujourd'hui, et il serait certainement intéressant de comparer cette version de l'opération de croisement avec une version utilisant un algorithme d'affinage.

En plus de cet opérateur de croisement, nous avons utilisé deux opérateurs de mutation différents. Le premier opérateur de mutation est celui défini dans le premier paragraphe de cette section, et qui consiste à déplacer un nombre fixe de sommets choisis aléatoirement vers des parties sélectionnées aléatoirement. Cet opérateur de mutation fait beaucoup intervenir l'aléatoire dans son fonctionnement. Le second opérateur de mutation est construit sur le même principe que le premier, mais en diminuant cette part de hasard. En effet, dans cet opérateur, seuls les sommets à la frontière de plusieurs parties peuvent être sélectionnés. Puis, lorsqu'un de ces sommets est aléatoirement choisi, il est échangé avec un des sommets pris au hasard dans l'ensemble des sommets adjacents, et d'une autre partie que celui-ci.

Le tableau 3.1 présente une comparaison des coûts de coupe des partitions trouvées par notre algorithme évolutionnaire et le logiciel pMetis (*supra* A.1). Ces résultats montrent que si notre adaptation trouve des solutions de qualité acceptable pour la bissection, elles sont très rapidement mauvaises quand le nombre de parties augmente. Ce problème nous a dissuadés de continuer plus loin avec cet algorithme.

Graphe	k	Algo. évo.	pMetis
add20	2	723	725
add20	4	1809	1292
add20	8	2640	1907
add20	16	3223	2504
data	2	252	218
data	4	1129	480
data	8	2307	842
data	16	3566	1370

TAB. 3.1 – Partitions obtenues par notre adaptation de l'algorithme évolutionnaire au partitionnement contraint.

3.5 Métaheuristiques hybrides pour le partitionnement contraint

Cette section propose un état de l'art des adaptations existantes de métaheuristiques pour le partitionnement de graphe contraint.

3.5.1 Présentation

Les métaheuristiques sont des méthodes qui permettent de trouver de très bon résultats si elles utilisent des heuristiques performantes et adaptées à leur problème. Pour le partitionnement de graphe, la plus performante d'entre elles semble sans conteste la méthode multi-niveaux, car elle associe rapidité et qualité des partitions trouvées. Cependant, de nombreuses autres métaheuristiques ont été utilisées pour résoudre ce problème, et parmi elles, les plus performantes sont souvent des méthodes hybrides ente une métaheuristique « classique » et une méthode multi-niveaux.

Cette section ne se veut pas exhaustive, mais propose au lecteur un certain nombre de méthodes hybrides performantes pour le partitionnement de graphe contraint.

Les métaheuristiques listées ci-dessous ont été sélectionnées car leurs résultats présentés sur le site de Chris Walshaw (*supra* A.3, *infra* annexe A.4) sont très bons en comparaison des autres méthodes :

– **ILS et multi-niveaux**. Cette première méthode hybride est à la fois très facile à mettre en place et performante. Elle combine une recherche itérative locale (ILS) avec un algorithme multi-niveaux. Ces dernières utilisent souvent des composantes aléatoires, que ce soit pour la contraction des sommets entre eux, ou l'affinage des partitions. Ainsi, changer la clef générant ces nombres aléatoires, permet de trouver une partition différente à partir du même graphe. Or, comme les outils de partitionnement classiques (*supra* A) sont sujets à des variations de coûts de coupe importantes, il est intéressant de sélectionner la meilleure des partitions trouvées après plusieurs itérations de ces logiciels. L'algorithme le plus simple à utiliser est la recherche itérative locale [LOR02] dont l'élément de perturbation est le changement de clef du générateur aléatoire, et dont la recherche locale utilise une méthode multi-niveaux de partitionnement. Une telle méthode hybride est entre autres utilisée dans [TOU99, LAN04a, WAL04]. Nous l'avons aussi utilisée pour des tests comparatifs (*infra* 7) ;
– **PROBE**, expansion de région et algorithme de Kernighan-Lin. La méthode PROBE, ou *Population Reinforced Optmization Based Exploration* en anglais, est présentée comme une nouvelle métaheuristique dans [BAR04, CHA07]. Basée sur une population d'individus, PROBE est inspirée des algorithmes génétiques. Cette nouvelle méthode est appliquée dans [CHA07] au partitionnement de graphe contraint. Pour cela, elle utilise comme heuristique un algorithme

d'expansion de région et comme méthode d'affinage un algorithme de type Kernighan-Lin. Les résultats obtenus par cette nouvelle approche sont très prometteurs, c'est pourquoi il nous a semblé important de l'insérer parmi les méthodes les plus performantes ;
- **Jostle-évolutionnaire**. L'algorithme évolutionnaire hybride de Alan Soper, Chris Walshaw et M. Cross est à la fois la méthode qui peut revendiquer le plus grand nombre de partitions présentes dans les bancs de tests du site de Chris Walshaw, et le temps d'exécution le plus long avec dans certains cas plus d'une semaine de calcul. Cette méthode est à la fois originale et intéressante, car permettant de trouver actuellement les meilleures partitions pour de nombreux graphes, c'est pourquoi nous l'avons décrite dans la sous-section suivante (*infra* 3.5.2).

Bien que le site de Chris Walshaw recense de nombreux outils de partitionnement, sa liste n'est pas exhaustive et d'autres outils méritent notre attention :
- l'algorithme multi-niveaux présenté dans [BAT99] utilise une méthode de recherche tabou lors des phases de partitionnement et d'affinage. Cette technique permet une plus grande homogénéité des coûts de coupes des partitions trouvées qu'avec une méthode multi-niveaux classique. C'est l'une des rares méthodes hybrides pour le partitionnement de graphe à utiliser une métaheuristique de voisinage, et aussi à présenter une méthode multi-niveaux utilisant une métaheuristique et non l'inverse ;
- l'algorithme hybride multi-niveaux/méthode évolutionnaire présentée dans [ALP96a] est l'une des premières approche hybrides rassemblant une métaheuristique et un algorithme multi-niveaux ;
- les méthodes spectrales ont elles aussi été hybridées avec des métaheuristiques, et plus particulièrement des algorithmes évolutionnaires [MAR05, MAR06] ;
- trois méthodes basées sur les algorithmes de colonies de fourmis ont retenu notre attention par les résultats qu'elles obtenaient :
  - la plus ancienne est présentée dans [LAN99]. Cette méthode, assez compliquée à mettre en place, utilise une méthode multi-niveaux dont l'étape de partitionnement est assurée par un algorithme évolutionnaire lui-même administrant plusieurs colonies de fourmis ;
  - la méthode dont nous venons de parler a été reprise et améliorée dans [ROB04, KOR04]. Les résultats présentés dans ces articles montrent que cette méthode fait probablement partie des meilleures adaptations des métaheuristiques au problème du partitionnement de graphe ;
  - l'algorithme présenté dans [COM06] se base uniquement sur un algorithme de colonies de fourmis. Ce n'est donc pas une méthode hybride, cependant, elle trouve, pour les graphes de petites tailles (quelques centaines de sommets), des partitions de coût de coupe très faibles. Pour les graphes de plus grande taille, une hybridation de cette méthode avec le multi-niveaux serait certainement intéressante.

3.5.2 Algorithme évolutionnaire multi-niveaux de Soper-Walshaw-Cross

L'algorithme évolutionnaire hybride de Alan Soper, Chris Walshaw et M. Cross surnommé Jostle-évolutionnaire, car l'algorithme évolutionnaire utilise la bibliothèque Jostle, est décrit dans [SOP00, SOP04b]. D'après les résultats publiés, ainsi que ceux des archives sur le partitionnement de graphe de Chris Walshaw (*infra* annexe A.4), cet algorithme est actuellement l'un des plus performants pour le partitionnement de graphe contraint. Il faut préciser que cette méthode n'est décrite que pour des graphes non pondérés.

Contrairement aux approches classiques des algorithmes évolutionnaires pour le partitionnement de graphe (*supra* 3.4), les individus de cet algorithme sont des variantes du graphe initial $G = (S, A)$. Les différents individus, c'est-à-dire les graphes, ne se distinguent pas par leurs structures, mais par la pondération de leurs arêtes. Cependant, l'objectif recherché reste celui du partitionnement de graphe contraint (*supra* 1.5), à savoir trouver une partition du graphe initial G en k parties minimisant le coût de coupe et respectant une balance de partitionnement bal_max. La qualité d'un individu doit donc être égale à celle d'une partition trouvée de manière déterministe à partir du graphe correspondant. Pour cela, les auteurs utilisent l'outil Jostle (*supra* A.1).

Remarque : bien que l'algorithme de Jostle soit stochastique, l'utilisation d'une même clef de son générateur aléatoire lors de plusieurs exécutions de cet algorithme sur une même machine permet de toujours trouver la même partition du graphe, si ce dernier est toujours présenté de façon identique.

Ainsi, l'évaluation d'un individu de la population coûte cher, même si un outil comme Jostle est rapide. Si l'on ajoute que les métaheuristiques ont généralement un temps d'exécution plus lent que les méthodes locales, il est facile de comprendre, avant même d'avoir présenté les opérateurs utilisés par la méthode évolutionnaire, que cet algorithme sera très lent. Son temps d'exécution est de plusieurs jours, mais peut atteindre plusieurs semaines [WAL04].

L'idée sous-jacente à cette méthode est de guider le choix de l'algorithme de partitionnement de Jostle. Pour cela, un biais positif est ajouté aux sommets du graphe en fonction de leur position dans la partition trouvée par Jostle. Le biais d'un sommet s est noté $biais(s)$. Le poids d'une arête $(s_1, s_2) \in A$ est alors égal à $poids(s_1, s_2) = 1 + biais(s_1) + biais(s_2)$. La valeur 1 représente le poids unitaire de l'arête. Le poids des sommets du graphe n'est pas modifié par le biais des sommets, car cela affecterait la balance de partitionnement. Cependant, l'application d'un biais sur le poids des arêtes du graphe va avoir pour effet de modifier la coupe de la partition trouvée par Jostle. En effet, cette coupe aura tendance à emprunter les arêtes dont le poids aura été abaissé par rapport au graphe initial au détriment des arêtes dont le poids aura été élevé.

Ainsi, la pondération des arêtes des graphes évolue dans les réels et non plus parmi les entiers, comme c'est le cas dans le reste de cette thèse. Cependant, le logiciel Jostle, comme toutes les méthodes multi-niveaux ou d'affinage, n'utilise habituellement que des pondérations entières. Ainsi, le logiciel Jostle a dû être modifié pour accepter les nombres réels. L'une des principales difficultés réside dans l'adaptation des tableaux de gains de l'implémentation de Fiduccia-Mattheyses (*supra* 2.5.1) au cas réel. En effet, à chaque gain, qui est une valeur entière, correspond une case du tableau. Les valeurs réelles des gains vont être discrétisées afin de pouvoir continuer à utiliser le tableau de gains. Soit un entier n, un réel $x \in [x - 0,5; x + 05]$ sera rangé dans le tableau de gain à la case de gain n.

Les opérateurs de croisement et de mutation consistent à faire évoluer le biais des sommets.

L'opérateur de croisement prend au moins deux parents pour créer un enfant. Les sommets qui sont à la frontière de la coupe des partitions de tous les parents se voient appliquer un biais tiré avec une probabilité uniforme dans $[0; 0,01]$. Un sommet à la frontière de la coupe d'une partition est un sommet faisant partie d'au moins une arête coupée par la partition trouvée par Jostle. Les autres sommets se voient affecter un biais d'une valeur de $0,1$ ajoutée à un nombre aléatoire choisi dans $[0; 0,01]$. Les auteurs semblent avoir trouvé ces valeurs de manière empirique. La figure 3.6 illustre cet opérateur de croisement dans le cas de deux parents : 3.6(a) et 3.6(b). Le graphe enfant est représenté figure 3.6(c). Dans celui-ci, les sommets entourés sont les sommets de biais le plus faible. Ces deux sommets grisés sont problématiques. En effet ceux-ci peuvent fausser le calcul de Jostle pour obtenir la partition de coût le plus faible, 3.6(d). Ils représentent la limite de cet opérateur de croisement.

L'opérateur de mutation consiste à déterminer pour la partition d'un individu, l'ensemble des sommets qui sont une distance d'au plus deux sommets d'une arête coupée par cette partition. C'est-à-dire, pour qu'un sommet appartienne à cet ensemble de voisinage, il faut au moins qu'il soit dans le voisinage d'un sommet lui-même dans le voisinage d'un sommet appartenant à une arête coupée par la partition. Puis, à chaque sommet de cet ensemble de voisinage, est associé un biais tiré avec une

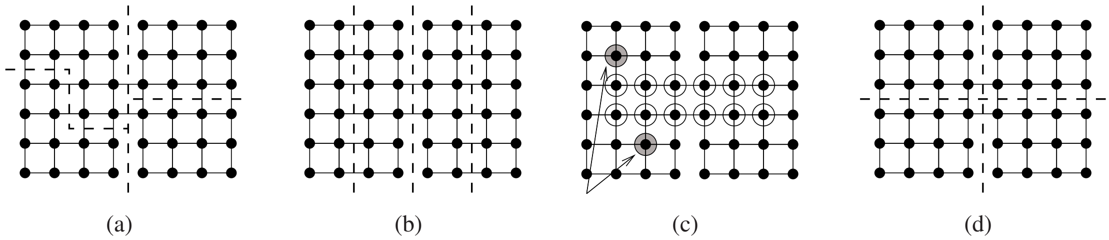


FIG. 3.6 – Illustration de l'opérateur de croisement de l'algorithme Jostle-évolutionnaire : 3.6(a) et 3.6(b) sont les 4-partitions trouvées par Jostle des deux graphes parents ; le graphe 3.6(c) est l'enfant créé, ses sommets entourés sont ceux dont le biais sera le plus petit ; le graphe 3.6(d) représente la meilleure partition possible.

probabilité uniforme dans $[0; 0, 01]$. Aux autres sommets du graphe est appliqué un biais de valeur $2, 0$ auquel s'ajoute un tirage dans $[0; 0, 01]$.

3.6 Conclusion

Ce chapitre a débuté par une section d'introduction sur les métaheuristiques, puis a continué dans les sections suivantes, par la présentation de trois métaheuristiques classiques. Chacune de ces sections est divisée en deux parties, la première étudiant la métaheuristique, la seconde décrivant notre adaptation de celle-ci au problème du partitionnement de graphe. Ainsi, nos deux adaptations du recuit simulé au partitionnement de graphe non contraint ont été présentées, puis notre adaptation des algorithmes de colonies de fourmis au problème du partitionnement non contraint a été introduit, enfin nos trois adaptations des algorithmes évolutionnaires au partitionnement de graphe contraint et non contraint ont été décrites. Ce chapitre s'est terminé par l'énumération des adaptations, issues de la littérature, les plus performantes de métaheuristiques pour le problème du partitionnement de graphe contraint.

Parmi les adaptations de ces trois métaheuristiques, la plus facile a été celle du recuit simulé. En effet, une modification élémentaire d'une partition est facile à mettre en œuvre, il suffit pour cela de déplacer des sommets de parties. La difficulté a été plus importante pour les adaptations des algorithmes évolutionnaires, principalement à cause de l'opérateur de croisement. Ce dernier c'est révélé compliqué à mettre en œuvre, car il repose sur la comparaison entre deux individus, or il est coûteux de comparer deux partitions. Enfin, l'adaptation qui a nécessité le plus de changements par rapport à la méthode initiale, a été celle des algorithmes de colonies de fourmis. Dans ce dernier cas, nous avons proposé un algorithme mettant en compétition plusieurs colonies pour le partage du graphe.

Chapitre 4

Élaboration d'une nouvelle métaheuristique : la méthode de fusion-fission

À l'origine, la méthode de fusion-fission a été créée spécialement pour résoudre le problème du découpage de l'espace aérien européen (*infra* 5). Ce problème, qui est l'application principale de cette thèse, est un problème de partitionnement non contraint. La création d'une nouvelle méthode pour la résolution du partitionnement de graphe non contraint est apparue intéressante après avoir testé différentes techniques classiques de partitionnement de graphe (*supra* 2 et 2.6) et plusieurs métaheuriques sur ce problème (*supra* 3).

La méthode de fusion-fission présentée dans ce chapitre est la contribution majeure de cette thèse. Cette méthode a été baptisée fusion-fission car le modèle qui a servi de base à son élaboration est celui de la physique nucléaire et plus particulièrement des fusions et fissions d'atomes.

La section 4.1 introduira les notions de physique nucléaire utilisées par la fusion-fission. La section 4.2 présentera les bases de la fusion-fission et les caractéristiques principales de la méthode. Puis les sections 4.3 et 4.4 présenteront respectivement l'application de cette méthode aux problèmes du partitionnement non contraint puis contraint. Comme nous le verrons, ces deux problèmes de partitionnement étant différents, les deux algorithmes de fusion-fission les résolvant sont différents, tout en gardant en commun les caractéristiques présentées en section 4.2. La section 4.5 sera l'objet d'une discussion sur les perspective futures de la fusion-fission. Enfin, la section 4.6 conclura la présentation de cette nouvelle méthode.

4.1 Quelques notions de physique nucléaire

Le but de cette section est d'exposer de manière simplifiée les notions de physique nucléaire qui nous ont servi à élaborer le modèle de fusion-fission. Nous ne prétendons en aucun cas être spécialistes en physique nucléaire. Le livre du professeur Daniel Blanc sur la physique nucléaire [BLA03] et le site internet[1] de David Calvet, permettent, pour ceux qui le souhaiteraient, d'aborder ce domaine de manière simple et claire.

L'atome est la plus petite partie d'un corps pur. Il est formé d'électrons et de nucléons. Les nucléons forment le noyau atomique et les électrons gravitent autour. Il existe deux sortes de nucléons,

[1] « Voyage au cœur de la matière... », http ://voyage.in2p3.fr/index.html

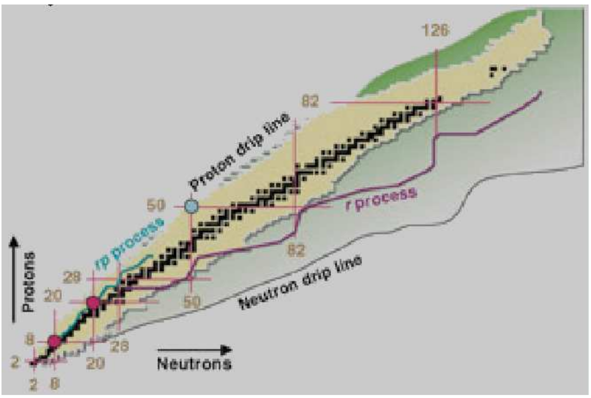


FIG. 4.1 – Diagramme de stabilité des noyaux en fonction de N et de Z. Carrés noirs : noyaux stables.

les protons de charge positive et les neutrons de charge neutre. Un atome contient donc trois sortes de particules : des électrons de charge négative, des protons et des neutrons. Le noyau atomique se situe au centre de l'atome, il est mille fois plus petit que ce dernier, mais contient presque la totalité de sa masse. La cohésion du noyau atomique est assurée par l'interaction forte, qui attire les nucléons entre eux et empêche ainsi les protons de se repousser.

Les atomes sont classés en fonction de leur nombre atomique Z dans le tableau périodique des éléments. Z est égal au nombre d'électrons, lui-même égal au nombre de protons d'un atome. Les isotopes sont des atomes qui ont le même nombre de protons Z, mais un nombre de neutrons N différent. A est le nombre de masse, c'est le nombre de nucléons. Un isotope est dit stable si son noyau n'est pas radioactif, sinon il est dit instable. Les isotopes stables sont représentés en noir dans la figure 4.1. Il y a aussi des valeurs « magiques » de N et de Z pour lesquelles la stabilité du noyau est maximale. Le symbole du noyau est $^A_Z M$ où M est le symbole de l'élément. Précisons deux autres notations : un neutron seul s'écrit $^1_0 n$, et un proton seul, $^1_1 p$.

À l'heure actuelle, nous connaissons quatre interactions. Elles interviennent dans tous les phénomènes de l'univers. La gravitation est l'interaction la moins intense, c'est une force attractive entre plusieurs corps qui dépend de leur masse. L'interaction électromagnétique est une force répulsive ou attractive qui agit sur des éléments possédant une charge électrique. L'interaction faible agit sur toutes les particules. L'interaction forte est une force de cohésion attractive très intense entre les quarks[2].

Certains atomes ont un noyau instable qui émet des rayonnements. On dit alors que ces atomes sont *radioactifs*. La radioactivité est la désintégration spontanée des noyaux. Ce phénomène est tout à fait naturel. Les interactions fortes, faibles et électromagnétiques sont à l'origine de différentes formes de radioactivité, les radio-activités alpha, béta et gamma.

Lors de la *fission* d'un atome, son noyau se scinde en deux fragments, en même temps que sont éjectés plusieurs neutrons. Les deux atomes restant sont en général radio-actifs. Ce qui veut dire dans le cas de la radio-activité alpha que chacun de ces atomes va se stabiliser en se séparant de deux protons et deux neutrons. Dans le cas de la radio-activité gamma, chaque atome va libérer de l'énergie sous la forme d'un photon. Dans le cas de la radio-activité béta, dans chaque atome, un neutron va se transformer en proton ou inversement.

Un atome peut fissionner soit de manière spontanée si son noyau est trop lourd, soit parce qu'il a été heurté par un neutron. Ainsi, les neutrons émis lors de la fission peuvent engendrer d'autres fissions, ce qui se traduit par une réaction en chaîne. C'est ce processus qui est utilisé dans les réacteurs

[2]En particulier, les nucléons sont constitués de trois quarks.

nucléaires, car la fission s'accompagne de libération d'énergie. Cependant, ce processus ne devient important que lorsque l'on est en présence de noyaux lourds, *i.e.* lorsque ces noyaux comportent un grand nombre de nucléons. Exemple de fission : l'uranium 235 sous l'impact d'un neutron d'énergie suffisante se transforme en uranium 236 hautement instable, qui se scinde :

$$
\begin{array}{l}
{}^{235}_{92}U +^1_0 n \rightarrow {}^{142}_{56}Ba +^{92}_{36} Kr + 2^1_0 n \\
{}^{235}_{92}U +^1_0 n \rightarrow {}^{134}_{54}Xe +^{90}_{38} Sr + 3^1_0 n
\end{array}
$$

La *fusion* de deux atomes apparaît lorsque les noyaux de ces deux atomes sont suffisamment proches l'un de l'autre pour fusionner, c'est-à-dire pour former un unique noyau. Pour fusionner, les atomes doivent être dans un milieu très chaud appelé plasma thermique. Ils auront alors des vitesses suffisamment élevées pour pouvoir fusionner avant d'être séparés par la force électromagnétique répulsive. Cette répulsion qui empêche habituellement les noyaux de fusionner, est due aux charges électriques positives des noyaux. Quand aucun état stable n'existe, il n'est pas toujours possible de provoquer la fusion de deux noyaux. Par exemple, deux atomes d'hélium stables ou deux atomes de fer stables ne peuvent pas fusionner ensemble. Mais voici quelques exemples de réactions de fusion existantes :

$$
\begin{array}{l}
{}^2_1H+^3_1H \rightarrow {}^4_2H_e +^1_0 n \\
{}^2_1H+^2_1H \rightarrow {}^3_2H_e +^1_0 n \\
{}^2_1H+^2_1H \rightarrow {}^3_1H +^1_1 p
\end{array}
$$

Nous pouvons voir, à travers cet exemple, que deux configurations identiques d'atomes peuvent engendrer des réactions de fusion différentes. La fusion de noyaux légers dégage d'énormes quantités d'énergie provenant de l'attraction entre les nucléons due à l'interaction forte. Lorsque de petits noyaux fusionnent, le noyau résultant se retrouve dans un état instable et doit revenir à un état stable d'énergie plus faible, en éjectant une ou plusieurs particules. C'est ce que nous avons pu constater avec les exemples de fusions d'hydrogène ci-dessus. Ainsi, les produits de la réaction de fusion ne sont pas radioactifs, et ce sont des éléments stables.

La masse d'un atome est inférieure à la somme des masses de ses nucléons isolés. Cette différence de masse correspond à la quantité d'énergie qu'il faudrait fournir au noyau pour le dissocier en ses constituants séparés. On appelle cette quantité l'énergie de liaison du noyau. La figure 4.2 donne la variation de l'énergie moyenne de liaison par nucléon en fonction du nombre de masse A. Cette courbe est appelée courbe d'Aston. On remarque sur cette figure que l'atome de fer est de stabilité maximale.

4.2 Les bases de la fusion-fission

Comme dans le cas de nombreuses métaheuristiques (*supra* 3), la fusion-fission s'inspire du monde réel. Plus particulièrement, la fusion-fission repose sur les notions de physique nucléaire présentées dans la section précédente (*supra* 4.1).

D'une manière générale, les métaheuristiques peuvent être considérées comme des entités propres, qui évoluent dans des mondes spécialement forgés à leur intention. Les métaheuristiques répondent uniquement aux propriétés des ces mondes et ceux-ci n'existent que pour elles. Chacun de ces mondes est basé sur plusieurs observations du monde réel. Ainsi, les algorithmes évolutionnaires reposent sur les observations de Darwin et sur son modèle d'évolution, mais ne tient pas compte de l'arrivée de prédateurs ou d'une maladie. Les colonies de fourmis sont basées sur l'observation du comportement

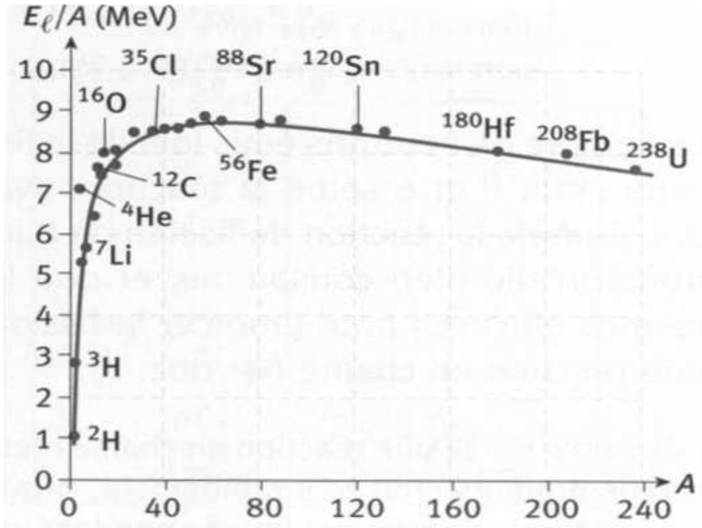


FIG. 4.2 – Courbe d'Aston. Présente, pour chaque atome, son énergie moyenne de liaison par nucléon.

collectif de celles-ci, hors interactions avec les menaces animales ou climatiques. De même, le recuit simulé est basé sur l'évolution de la structure du métal sans tenir compte du façonnage du forgeron sur celui-ci.

De plus, ces mondes sont limités pour pouvoir s'adapter facilement à une programmation informatique. Ainsi, de nombreuses propriétés du monde réel ne sont pas présentes dans les métaheuristiques, du moins dans leurs formes classiques. Par exemple, les algorithmes évolutionnaires n'incorporent pas la notion d'âge et de vieillissement des individus. Il n'y a pas de cycle de reproduction des fourmis dans les algorithmes du même nom, ni d'agrandissement de leur territoire. Le refroidissement du métal lors d'un recuit est uniforme sur tout le matériau, et ce quelle que soit la courbe de décroissance de celui-ci.

Ces mondes intègrent des propriétés exacerbées du monde réel, ou même certaines n'existant pas. Dans le cas des algorithmes évolutionnaires, il n'existe pas dans le monde réel de populations où un individu vit des centaines de fois plus longtemps que l'espérance de vie de la population, tout en continuant à se reproduire beaucoup plus que le reste de la population. Les fourmis ont une longévité et une endurance infinie. Dans un recuit simulé, la température décroît dans tout le matériau en suivant parfaitement une courbe de température, ce qui semble physiquement irréaliste.

Nous allons maintenant créer un nouvel univers imaginatif dans lequel évoluera la méthode de fusion-fission, en y incorporant des limites et des lois. Cependant, comme dans les cas des autres métaheuristiques, celles-ci ne seront pas toujours en corrélation avec celles du monde réel. Il nous a semblé que la manière la plus simple de créer un nouvel univers était de partir du « big bang », point zéro du temps tel que nous le percevons. Dès lors, faisons table rase de tout ce qui existe et tâchons de recréer un monde pour notre problème en partant du big bang.

Tout d'abord, rappelons que nous cherchons à résoudre un problème d'optimisation combinatoire Π avec un espace de solutions E et un espace de solutions admissibles E_A dans lequel on cherche à minimiser une fonction objectif f. Chaque solution du problème est entièrement caractérisée par un ensemble de n paramètres élémentaires : $\{p_1, \dots, p_n\}$. L'objectif de toute méthode d'optimisation est de trouver un paramétrage définissant un individu minimisant le plus possible f. Ainsi, comme toutes les autres méthodes d'optimisation, la fusion-fission va faire évoluer progressivement ces paramètres, à travers leur valeur, leur organisation, *etc*.

Revenons au Big Bang. Celui-ci représente l'état initial de notre optimisation. Dans notre processus d'optimisation, il s'agit d'une solution prise au hasard dans l'espace des solutions admissibles. Cette première solution est donc supposée chaotique. Considérons chacun des paramètres de cette solution comme étant un nucléon. L'univers ainsi créé est de petite taille comparé au notre puisque, dans la majorité des problèmes, il va comporter moins d'un million de nucléons, *i.e.* le nombre de paramètres permettant de définir une solution admissible du problème d'optimisation est inférieur à un million.

Juste après le Big Bang, l'univers est très dense, et les nucléons fusionnent pour créer des atomes. Puis certains atomes récemment formés vont être détruits par des nucléons libres, ou encore des atomes instables vont naturellement se diviser. Toutes ces fusions et fissions peuvent libérer des nucléons qui vont eux-mêmes provoquer d'autres fusions ou fissions. Comme nous l'avons vu, les atomes précédant le fer dans le tableau périodique des éléments, libèrent de l'énergie en fusionnant et ceux le succédant en libèrent par la fission. L'état chaotique fait qu'il va aussi y avoir fission de petits atomes et fusion de gros atomes. Ce processus ne prend pas en compte la création de molécules, c'est-à-dire d'assemblages d'atomes. L'optimum de l'organisation atomique est atteint lorsque l'univers possède un maximum d'atomes de fer.

Durant le processus de fusion et de fission des atomes, plusieurs paliers vont être successivement franchis avant d'arriver à une configuration optimale possédant un maximum d'atomes de fer. Si ces paliers ne sont pas correctement franchis, la configuration optimale aura très peu de chances d'être trouvée. Les paliers correspondent aux configurations où la grande majorité des atomes comportent un certain nombre de nucléons. Ainsi, le premier palier correspondant à l'état initial est celui où tous les atomes sont des atomes hydrogène (il ne possède qu'un seul nucléon). Puis les paliers se succèdent, hélium, lithium, béryllium, etc, jusqu'au fer. On peut aussi imaginer que ces paliers ne se succèdent pas linéairement et de façon croissante, et qu'un palier de niveau supérieur peut retomber après plusieurs fissions à un palier inférieur.

Pour reprendre l'approche empruntée au début de cette section, le monde que nous considérons est celui d'un nuage de nucléons. Celui-ci est soumis à une température et une pression très fortes, ce qui fait que les nucléons vont avoir de grandes chances d'entrer en collision. C'est la fusion de ces nucléons entre eux, puis des atomes résultants, qui va permettre d'atteindre l'équilibre du système. Ce dernier est atteint lorsque tous les atomes du nuage sont du fer. La fission est utilisée afin de faire éclater les atomes trop gros, ou encore ceux déséquilibrés (dont les caractéristiques ne respectent pas les contraintes du problème). Les limites de ce monde imaginaire sont la taille du problème, qui va directement influencer le nombre de nucléons. Nous supposerons que température et pression seront toujours suffisamment élevées pour permettre un grand nombre de fusions comme de fissions, ce ne seront pas des paramètres déterminants du système. Des lois peuvent être créées et un processus d'apprentissage peut être utilisé pour les faire évoluer. Ces lois régulent les mécanismes de fusion comme de fission. Elles peuvent déterminer des actions obligatoires (par exemple, la fission d'un atome trop gros), des actions comportant une partie probabiliste (l'éjection de nucléons et leur nombre) ou encore des actions interdites (comme la fission d'un atome trop petit).

Les métaheuristiques sont en générales conçues autour de deux tâches classiques : l'*intensification* et la *diversification*. L'intensification permet de « descendre » vers un optimum local du problème, c'est-à-dire, dans un puits énergétique, alors que la diversification permet à l'algorithme de s'extraire du puits énergétique pour en chercher un nouveau plus profond. Dans la méthode de fusion-fission, on peut considérer que la tâche d'intensification est en grande partie remplie par la fusion, alors que celle de diversification est exécutée par la fission. La fusion a pour but de rassembler les paramètres entre eux, en écartant les paramètres trop « éloignés ». Son but est donc de créer une solution plus fine du problème en s'aidant des paramètres les plus prometteurs. À l'inverse, la fission va disperser les paramètres. Une nouvelle disposition de ceux-ci va être créée dans le but de chercher à explorer un autre espace de solutions.

Malgré nos efforts, il est encore difficile de décrire précisément une métaheuristique de fusion-fission. En effet, cette méthode n'en est encore qu'à ses prémisses, et son adaptation à différents problèmes d'optimisation combinatoire est encore très pauvre. Cependant, si cela peut paraître une lacune, c'est aussi une richesse, car cette méthode a encore de multiples horizons pour évoluer et nous connaissons par l'exemple des algorithmes évolutionnaire, la grande variété des évolutions possibles d'une métaheuristique.

4.3 Application au partitionnement non contraint

L'application de la méthode de fusion-fission au partitionnement non contraint (*supra* 1.6) a été la première application de la fusion-fission. Cette application a fait l'objet de plusieurs publications [BIC04b, BIC06a, BIC07].

Entre ces différentes publications, l'algorithme de fusion-fission a évolué. La version actuelle de cet algorithme est présentée dans [BIC07]. Cependant, il nous a paru intéressant de mentionner certaines adaptations utilisées précédemment.

4.3.1 Mise en place de la méthode

Pour résoudre le problème de partitionnement non contraint, nous avons cherché une méthode proche d'une métaheuristique permettant de regrouper des sommets dans les parties d'une partition. Dans le processus de fusion-fission présenté précédemment (*supra* 4.2), les parties d'une partition sont comparées à des atomes, les sommets du graphe à des nucléons et une partition à une configuration d'un palier de la fusion-fission. Soit k le nombre de parties recherchées. Alors que les méthodes classiques comme le recuit simulé agissent en déplaçant les sommets du graphe dans un nombre fixe de parties, le but de la fusion-fission est de pouvoir déplacer un nombre quelconque de sommets vers un nombre de parties non fixe. En effet, dans certains cas, une partition d'énergie significativement plus basse peut être trouvée pour un nombre de parties différent de celui recherché.

Nous avons vu précédemment, en section 4.1, que la fusion comme la fission font interagir les atomes entre eux et plus particulièrement les nucléons entre eux. Le cœur des atomes est composé de nucléons, donc en faisant abstraction des électrons gravitant autour du noyau, on peut considérer que les atomes sont seulement des ensembles de nucléons. Pour simplifier encore le modèle, nous pouvons ne pas tenir compte de la différence entre neutrons et protons dans les réactions, seule leur qualité de nucléons va nous intéresser.

La température joue un rôle important dans les mécanismes de fusion-fission. Lorsque la température diminue, l'état des particules devient plus stable, et l'on pourrait imaginer que les atomes convergent vers un état où leur énergie de liaison est maximale, c'est-à-dire vers l'atome de fer comme le montre la figure 4.2. À l'inverse, lorsque la température augmente, les atomes peuvent devenir très gros, ou très petits, selon qu'un processus de fusion ou de fission en chaîne a commencé.

Les réactions de fusion et de fission obéissent à des règles bien précises, mais facilement généralisables. Lors d'une fusion, deux atomes peuvent s'assembler en libérant ou non un petit nombre de nucléons. Ces nucléons une fois libérés viennent se rattacher à d'autres atomes dont certains peuvent ainsi devenir radioactifs. Lors d'une fission, un atome se scinde en deux parties en pouvant libérer ou non un petit nombre de nucléons. Si leur énergie est suffisante, ces nucléons peuvent à leur tour engendrer la fission d'autres atomes voisins, et ainsi engendrer une fission en chaîne. Sinon, ils se rattachent à d'autres atomes par fusion. Ces procédures de fusion et de fission sont modélisées en formulant des règles auxquelles elles doivent obéir.

Dans la nature, les atomes de fer possèdent l'énergie moyenne de liaison par nucléon la plus forte. Nous allons considérer, dans notre modèle, qu'un état possédant k atomes correspond à l'état où l'énergie moyenne de liaison par nucléon est la plus forte. En effet, puisque le nombre de sommets d'un graphe est fixe et qu'ainsi le nombre de nucléons l'est aussi, seul le nombre d'atomes peut permettre de faire varier le nombre de nucléons par atomes.

L'état de plus basse énergie est conservé pendant l'optimisation afin de comparer l'état courant avec celui-ci. C'est ce meilleur état atteint qui sera renvoyé à la fin de l'optimisation. La fusion-fission, de par sa nature, va changer le nombre d'atomes des différents états. Pourtant c'est une partition en k parties d'énergie minimale qui est recherchée. La méthode de fusion-fission va chercher une telle partition en faisant varier le nombre de parties autour de k. Ainsi, l'un des avantages de la fusion-fission est qu'elle va donner des résultats satisfaisants pour des partitions de nombres de parties différentes.

Cependant la fonction de coût doit pénaliser les configurations d'états dans lesquelles le nombre d'atomes est éloigné du nombre k. La fonction de coût doit imiter la courbe inverse de la figure 4.2, car l'on cherche à minimiser l'énergie et non à la maximiser comme c'est le cas de l'énergie de liaison. Pour adapter la fonction de coût, il suffit de la multiplier par une fonction proportionnelle à l'écart entre le nombre d'atomes d'un état et k.

Nous distinguons deux étapes dans notre méthode. La première étape, l'initialisation, va partir d'un état qui n'est pas solution du problème. Dans cet état ne sont présents que les nucléons, et pas encore les atomes. Cette initialisation a pour but de créer un état ne laissant plus aucun nucléon seul, et d'énergie inférieure à une énergie seuil. L'initialisation est suivie de l'optimisation proprement dite. Celle-ci partira de l'ensemble d'atomes issus de l'initialisation et fera évoluer l'état vers un état de plus basse énergie. Pour cela, l'état de départ de l'optimisation sera considéré comme un plasma thermique à très haute température. La température descendra progressivement, en même temps que se réaliseront les fusions et fissions entre atomes. Quand la température sera trop basse, l'étape d'optimisation sera réitérée à partir du meilleur état précédemment trouvé.

Mais avant de décrire ces deux étapes, nous devons expliquer le mécanisme de règle dont nous avons parlé précédemment et qui sera utilisé par chacune de ces étapes.

4.3.2 Le mécanisme de règle

Le mécanisme de règle présenté dans cette sous-section a été construit de manière empirique. Il en résulte que ce n'est pas le seul mécanisme qui puisse fonctionner avec notre méthode. Nous avons créé ce mécanisme car il nous semblait à la fois être proche du modèle de fusion-fission nucléaire que nous cherchons à imiter, et bien adapté à notre problème.

Le mécanisme de règle sert à décrire les fusions et les fissions, et en particulier renvoie le nombre de nucléons éjectés par elles. Dans le mécanisme de règle, les réactions sont classées différemment selon que ce sont des fissions ou des fusions. Dans le cas d'une fission, une règle différente est utilisée en fonction du nombre de nucléons présents dans l'atome. Pour une fusion, la règle est indexée par les nombres de nucléons des deux atomes en présence. Les règles sont des tableaux de listes de probabilités. Pour chacune de ces listes, la somme des probabilités vaut un. Chaque probabilité est strictement comprise entre zéro et un. Chaque probabilité de la liste correspond au nombre de nucléons indépendants qui vont apparaître en plus de l'atome fusionné ou des deux nouveaux atomes résultant de la fission. De plus, ces règles possèdent un mécanisme d'apprentissage. Lorsqu'une des

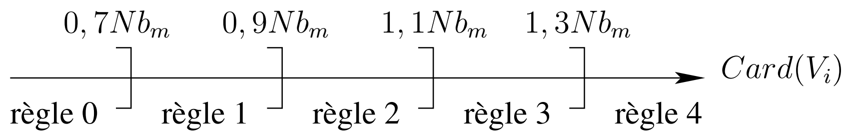


FIG. 4.3 – Une règle est choisie en fonction du nombre de sommets de la partie et du nombre moyen de sommets dans une partie, $Nb_m = \frac{Card(S)}{k}$.

probabilités d'une règle s'est révélée satisfaisante, par exemple si le nouvel état atteint grâce à elle est moins énergétique, cette probabilité est augmentée au détriment des autres probabilités de cette règle. La règle sera alors « acceptée positivement ». A l'inverse, si cette probabilité ne s'est pas montrée satisfaisante, elle sera pénalisée au profit des autres probabilités de cette règle. La règle sera alors « acceptée négativement ». Il faut cependant que toutes les probabilités soient strictement positives, pour toujours laisser la possibilité d'avoir un nombre de nucléons différents.

Prenons un exemple. Soit la règle permettant la séparation d'au plus 3 nucléons :

$$\begin{cases} fusion : & \{p_{u0}; p_{u1}; p_{u2}; p_{u3}\} \ , \\ fission : & \{p_{i0}; p_{i1}; p_{i2}; p_{i3}\} \ . \end{cases}$$

Nous avons $\forall i,\ 0 < p_i < 1$ et $\sum_i p_i = 1$. Dans cette règle, la probabilité qu'une fusion ne se sépare d'aucun nucléon est p_{u0}, et p_{i0} lors d'une fission. Et la probabilité de se séparer de trois nucléons est p_{u3} pour une fusion et de p_{i3} pour une fission. Dans le cas où une fusion est réalisée et rejette un nucléon, et où l'état obtenu est de plus basse énergie que le précédent, la nouvelle règle pour la fusion sera :

$$\left\{ p_{u0} - \frac{pas\_de\_r\grave{e}gle}{3}; p_{u1} + pas\_de\_r\grave{e}gle; p_{u2} - \frac{pas\_de\_r\grave{e}gle}{3}; p_{u3} - \frac{pas\_de\_r\grave{e}gle}{3} \right\} \ ,$$

avec $pas\_de\_r\grave{e}gle$ un paramètre de variation de la règle pour que celle-ci apprenne au fur et à mesure. Dans le cas où l'énergie serait supérieure, la nouvelle règle pour la fusion serait le même ensemble que précédemment en remplaçant les additions par des soustractions et inversement. Pour la fission, le même mécanisme de mise à jour de la liste est utilisé.

Une variante de ce mécanisme de règle est présenté dans [BIC07]. Dans celle-ci, à chaque probabilité de fusion p_u est associé un pourcentage des nucléons dont la partie devra se séparer. Ce pourcentage est contenu dans le tableau $pnucl\_\acute{e}ject\acute{e}s$. Il en est de même pour les probabilités de fission. Supposons qu'à la probabilité p_{u2} est associé le pourcentage c_{u2}. Alors, lorsque p_{u2} sera sélectionné, c_{u2} pour cent des nucléons de la partie seront séparés d'elle pour être intégrés ailleurs. De plus, pour la fusion comme pour la fission, les règles sont regroupées en 5 listes de probabilités. La première règle est sélectionnée quand la partie a un nombre de sommets moins de $0,7$ fois le nombre de sommets moyens d'une partie, la seconde quand elle a un nombre de sommets entre $0,7$ et $0,9$ fois le nombre de sommets moyens d'une partie, et ainsi de suite comme le présente la figure 4.3. Cette astuce permet de réduire le nombre total de règles à 10, et ainsi d'avoir un mécanisme d'apprentissage plus rapide.

4.3.3 Fonction d'adaptation de la fonction de coût

Comme nous l'avons vu dans la sous-section 4.3.1, le nombre de parties de la partition varie au cours de l'optimisation. Or les fonctions de coût (*supra* 1.4) ont une valeur qui dépend intrinsèquement de ce nombre de parties. Pour ces fonctions objectifs, la partition de coût le plus faible est celle qui ne comporte qu'une partie, c'est-à-dire, la partition est égale au graphe. Or, sans adaptation de la fonction de coût, notre algorithme converge rapidement vers ce minimum. Pour éviter cette convergence il suffit d'utiliser une fonction de correction modifiant la valeur de la fonction objectif.

La fonction d'adaptation doit être neutre pour les partitions dont le nombre de parties est proche de k, mais doit être très pénalisante lorsque ce nombre s'éloigne trop de k. La fonction de correction suivante a été trouvée de façon empirique pour le problème du découpage de l'espace aérien européen (*infra* 5) :

$$correction(partition) = \left(\frac{k - card(partition)}{3}\right)^3 + 1\ .$$

La fonction de correction doit être adaptée au problème que l'on cherche à résoudre, c'est pourquoi pour une adaptation à un autre problème, il pourrait être utile de modifier celle-ci.

Ainsi, si l'on désigne par f la fonction objectif à minimiser, la fonction de coût adaptée pour la fusion-fission devient :

$$énergie(partition) = f(partition) * correction(partition)\ .$$

4.3.4 Le processus d'initialisation

Le processus d'initialisation peut être vu comme une version simplifiée du processus d'optimisation. L'initialisation démarre sur un état E_{init}, *i.e.* une partition, constituée uniquement de $Card(S)$ nucléons indépendants, chacun dans un atome. Prenons un état E_i de la matière à l'étape i du processus d'initialisation. À partir de cet état, un atome ou un nucléon est tiré au hasard. Une fonction de probabilité détermine la probabilité qu'il y ait fusion ou fission (*infra* 4.3.6). À partir de ce choix, une fusion ou une fission va être réalisée :

- si c'est une fusion (*infra* 4.3.7), un atome va être sélectionné en fonction de critères énergétiques et de distance au premier atome. Nous entendons par distance la somme des poids des arêtes entre deux atomes. La règle de fusion correspondant à la somme du nombre de nucléons des deux atomes est alors sélectionnée. Puis la fusion est réalisée. Elle peut libérer des nucléons qui sont alors fusionnés avec d'autres atomes ou nucléons ;
- si c'est une fission (*infra* 4.3.8), la règle correspondant au nombre de nucléons de l'atome est utilisée. En accord avec la règle correspondante, des nucléons peuvent être éjectés. Dans le cas de l'initialisation, nous considérons que leur énergie est trop faible pour engendrer d'autres fissions. Ces nucléons vont donc fusionner avec d'autres atomes.

Dans ces deux cas, le nombre d'atomes va avoir changé, et l'état suivant est obtenu. Puis ce processus d'initialisation est réitéré jusqu'à ce qu'un état ne possédant plus de nucléon isolé, un nombre d'atomes proche de k, et étant d'énergie inférieure à une énergie d'initialisation seuil, soit trouvé. L'initialisation se rapproche d'un mécanisme de fusion en chaîne, contrôlé par la possibilité de fission lorsque des atomes deviennent disproportionnés par rapport aux autres atomes. L'algorithme 17 présente le pseudo-code de la méthode d'initialisation.

4.3.5 Le processus d'optimisation

Le fonctionnement du processus d'optimisation reprend, en le complétant, celui de l'initialisation. Il part de la partition trouvée par l'algorithme d'initialisation. Comme dans le cas de l'initialisation (*supra* 4.3.4), l'optimisation se fait par itérations successives de transformations d'une partition. L'algorithme 18 présente le pseudo-code de ces processus. L'algorithme s'articule autour d'une boucle principale qui s'arrête lorsqu'une condition d'arrêt (par exemple un temps de calcul

Algorithme 17 Initialisation de l'adaptation de la fusion-fission au partitionnement non contraint.

```
Procédure InitialisationFusionFission(G = (S, A), règles)
    état ← partition de S en Card(S) parties
    tant que non fin_initialisation(état) faire
        atome ← choix de l'atome initial en fonction des règles
        si choix_init(atome, état) = fusion alors
            (état, règles, liste_nucléons) ← Fusion(atome, état, règles) ;
        sinon
            (état, règles, liste_nucléons) ← Fission(atome, état, règles) ;
        fin si
        pour tout nucléon ∈ liste_nucléons faire
            (état, règles) ← Fusion(nucléon, état, règles) ;
        fin pour
    fin tant que
    retourner (état, règles) ;
fin Procédure
```

minimum) est satisfaite. Avant le début de cette boucle, la structure des règles (*supra* 4.3.2) est initialisée pour la fusion comme pour la fission. Chaque passe dans la boucle principale commence par récupérer la meilleure partition précédemment obtenue pour l'utiliser en temps que partition courante. Une seconde boucle commence alors. Elle sera nommée boucle intérieure. Elle possède un nombre d'itérations fixes, nombre_de_passes. À chaque passage dans la boucle intérieure, une partie de la partition courante est sélectionnée aléatoirement. Une fonction de choix (*infra* 4.3.6) détermine alors s'il faut appliquer à cette partie une fusion (*infra* 4.3.7) ou une fission (*infra* 4.3.8). Puis, après que la partie ait été fusionnée avec une autre partie, ou qu'elle ait été divisée, la règle qui a permis la fusion ou la fission est mise à jour. Si la partition trouvée est de meilleure qualité, alors la meilleure partition trouvée est mise à jour. Puis les boucles intérieures, puis principales, peuvent itérer.

4.3.6 Fonctions de choix entre fusion et fission

Nous utilisons deux fonctions de choix pour déterminer si un atome $partie$ (un atome est une partie de la partition) subira une fission ou une fusion. La première fonction concerne le processus d'initialisation, et la seconde, le processus d'optimisation proprement dit.

De nombreuses fonctions de choix ont été testées pour déterminer si une partie d'une partition devait subir une fusion ou une fission. Nous n'en détaillerons que deux dans cette sous-section. Bien d'autres fonctions pevent être utilisées à la place de celles que nous présentons, cependant, parmi toutes celles que nous avons développées, ces deux fonctions sont celles qui nous ont semblé les plus efficaces.

Fonction de choix pour l'initialisation

La fonction de probabilité pour l'initialisation utilise deux valeurs communes à toute implémentation de la méthode de fusion-fission, le nombre d'atomes de l'état courant et le nombre de nucléons de l'atome $partie$. Si le nombre de nucléons d'un atome est grand proportionnellement au nombre d'atomes, nous voulons qu'il y ait une fission ; si ce nombre est petit, qu'il y ait une fusion ; et si ce nombre est médian, que les probabilités soient égales. Pour cela nous avons construit une fonction

Algorithme 18 Adaptation de la fusion-fission au partitionnement non contraint.

Procédure FUSIONFISSIONNONCONTRAINT($G = (S, A)$, partition_initiale)
  règles ← initialisation des règles
  meilleure_partition ← partition_initiale
  tant que non condition_d'arrêt **faire**
    partition ← meilleure_partition ;
    pour température = 1 **à** nombre_de_passes **faire**
      partie ← choix aléatoire d'une partie de partition
      si choix_fusion(partie, partition, température) **alors**
        partition ← FUSION(partie, partition, règles) ;
      sinon
        partition ← FISSION(partie, partition, règles) ;
      fin si
      règles $gets$ mise à jour des règles
      si énergie(partition) < énergie(meilleure_partition) **alors**
        meilleure_partition ← partition ;
      fin si
    fin pour
  fin tant que
fin Procédure

$h : [-1; 1]^2 \rightarrow R$ qui répond à cette problématique :

$$h(x, y) = \frac{\exp^{\gamma(x+y)} - 1}{\exp^{\gamma(x+y)} + 1}\,.$$

Cette fonction possède des points d'inflexion pour $h(x, -x) = 0,\ \forall x$. Le paramètre γ régule la pente de la courbe autour de ses points d'inflexion. Nous avons choisi cette fonction car elle permet une diminution rapide, puis progressive, des probabilités à partir de ses points d'inflexions (*cf* figure 4.5), et inversement au dessus de ceux-ci.

Cependant, cette fonction n'est pas une fonction de probabilité, puisque $Img(h) \neq [0; 1]$. Nous avons donc construit une fonction $t_z : Img(h) \rightarrow [0; 1]$, telle que :

$$t_z(z) = \frac{z + h\left(t_x(nb\_tot\_nucléons), t_y(nb\_tot\_nucléons)\right)}{2\,h\left(t_x(nb\_tot\_nucléons), t_y(nb\_tot\_nucléons)\right)}\,.$$

Pour les points d'inflexion, les probabilités sont égales puisque $\forall x,\ h(x, -x) = \frac{1}{2}$. De plus, lorsque le nombre d'atomes est négligeable devant celui des nucléons de $partie$, $t_z(h(x, y)) \longrightarrow 1$, et lorsque le nombre de nucléons de $partie$ est négligeable devant celui d'atomes, $t_z(h(x, y)) \longrightarrow 0$.

Nous connaissons le nombre maximal d'atomes pour l'initialisation, qui est égal au nombre de nucléons, $Card(S)$, et le nombre final d'atomes k que l'on veut obtenir. Le nombre minimal d'atomes est égal à 1. De même, nous connaissons le nombre maximal de nucléons par atome, et le nombre moyen de nucléons par atome :

$$Nb_m = \frac{Card(S)}{k}\,.$$

Le nombre minimal de nucléons dans un atome est 1. Pour pouvoir utiliser la fonction h, nous devons créer deux fonctions adaptant le nombre d'atomes pour t_x, et le nombre de nucléons pour t_y,

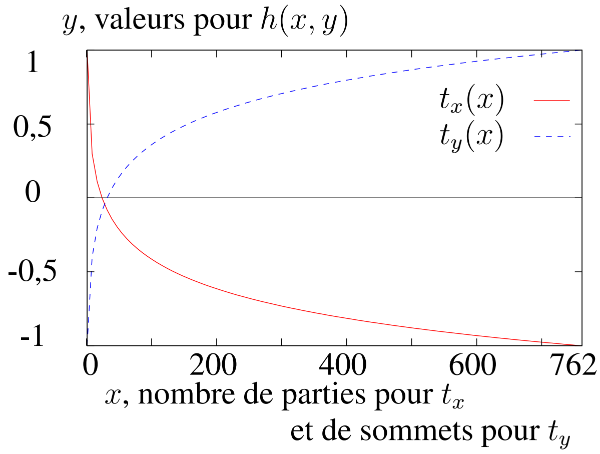


FIG. 4.4 – Courbes des fonctions $t_x(x)$ et $t_y(x)$.

au domaine de définition de h. Ces fonctions doivent respecter les contraintes suivantes : $t_x(k) = 0$ et $t_y(Nb_m) = 0$ afin que $h(t_x(k), t_y(Nb_m)) = \frac{1}{2}$, $t_x(Card(S)) = t_y(Card(S)) = 1$ et $t_x(1) = t_y(1) = -1$ pour ne pas sortir du domaine de définition de h. La figure 4.4 représente ces fonctions. Soit

$$t(x,v) = \frac{\log\big((Card(S) - 2v + 1)x + v^2 - Card(S)\big) - \log\big((v-1)(Card(S) - v)\big)}{\log(Card(S) - v) - \log(v-1)} \ .$$

Nous posons pour une partition P : $t_x(x) = -t(x, Card(P))$ et $t_y(y) = t(x,n)$.

Pour finir, nous avons défini la fonction de probabilité $choix\_atome\_init$ représentée figure 4.5 :

$$choix\_atome\_init(x,y) = t_z\left(h\left(t_x(x), t_y(y)\right)\right) \ .$$

Fonction de choix entre fusion et fission

Pour le processus d'optimisation, nous avons voulu une fonction de choix très simple et qui soit facilement généralisable. Nous avons donc abandonné les courbes avec points d'inflexion, et nous nous sommes concentrés sur un seul argument, le nombre de nucléons de l'atome. Nous avons laissé de côté le nombre d'atomes. Pourtant, durant l'optimisation, le nombre d'atomes devra osciller autour de k. Mais ce n'est pas à la fonction de choix d'influer directement sur le nombre d'atomes, mais à la fonction de coût.

Nous avons défini la fonction de probabilité $choix$ présentée figure 4.6 par :

$$choix(partie) = \begin{cases} 1 & si\ Card(partie) > Nb_m + \frac{1}{2\alpha(T)}\ , \\ 0 & si\ Card(partie) < Nb_m - \frac{1}{2\alpha(T)}\ , \\ \alpha(T) * (Card(partie) - Nb_m) + \frac{1}{2} & sinon\ , \end{cases}$$

où la fonction alpha dépend de la température T et des températures maximales (T_{max}) et minimales (T_{min}) du plasma. Cette fonction alpha s'écrit :

$$\alpha(T) = p\,\frac{nombre\_de\_passes - T}{nombre\_de\_passes} + q\ ,$$

avec p et q deux paramètres.

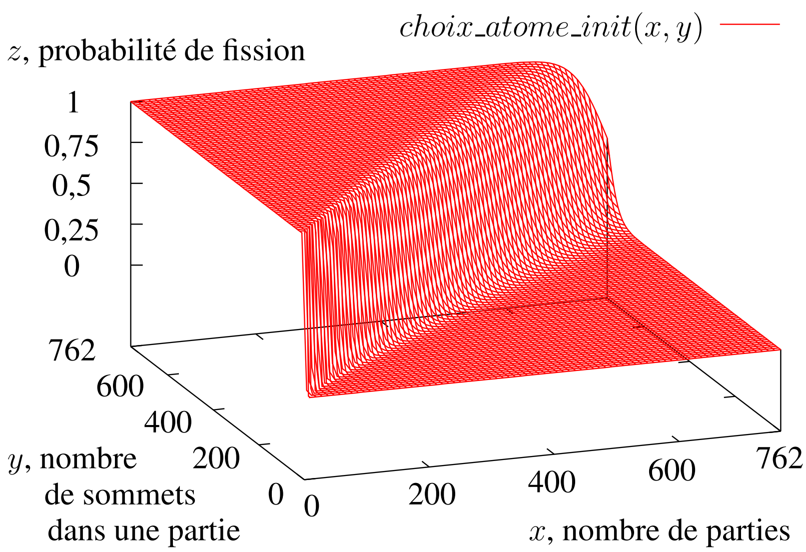


FIG. 4.5 – Courbe de $choix\_atome\_init(x, y)$.

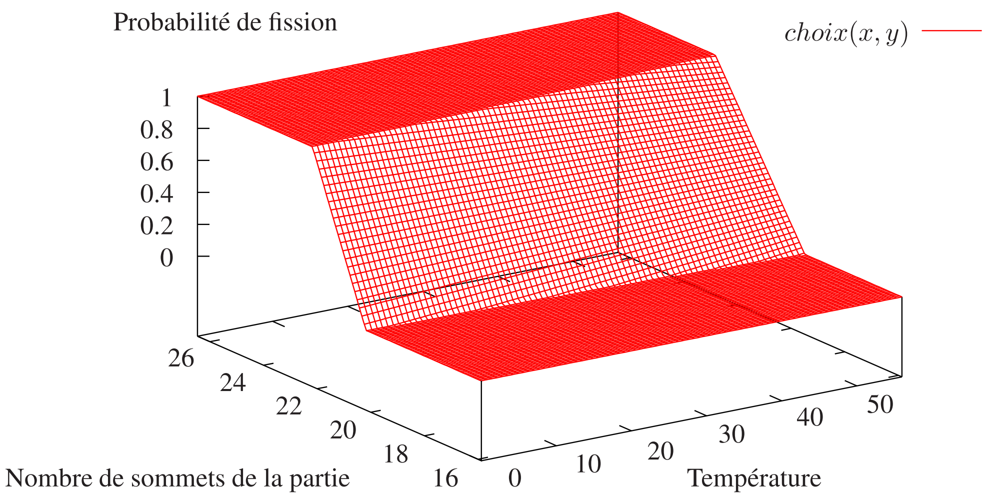


FIG. 4.6 – Courbe de la fonction de choix entre fusion et fission : $choix$.

La fonction $choix$ est une simple droite de pente $\alpha(T)$ entre $Nb_m - \frac{1}{2\alpha(T)}$ et $n + \frac{1}{2\alpha(T)}$, et telle que $choix(Nb_m) = \frac{1}{2}$. Ainsi, il y a autant de chance d'avoir une fusion ou une fission lorsque le nombre de nucléons de l'atome $partie$ est égal au nombre moyen de nucléons. Le choix entre fusion et fission est probabiliste : soit un réel $r \in [0; 1]$ tiré aléatoirement, s'il est plus grand que la valeur $choix(partie)$, alors il y aura fusion, sinon il y aura fission. Cependant, quand le nombre de nucléons de $partie$ est trop grand, $Card(partie) > Nb_m + \frac{1}{2\alpha(T)}$, il y a toujours fission, et quand il est trop petit, $Card(partie) < Nb_m + \frac{1}{2\alpha(T)}$, il y a toujours fusion.

La température est utilisée pour réguler le processus comme elle le fait lors d'une réaction nucléaire. Plus chaud est le plasma, plus grande est la probabilité d'avoir une fusion ou une fission. Le poids de la partie peut être utilisé à la place de son cardinal, mais l'utilisation du cardinal de la partie permet de contrôler la balance du nombre de sommets de la partition.

4.3.7 Fusion

L'algorithme de fusion a pour rôle d'agréger deux parties ensemble au sein d'une partition. L'algorithme 19 présente son pseudo-code. La fusion prend en paramètres une première partie sélectionnée aléatoirement au début de la boucle intérieure de l'algorithme de fusion-fission, ainsi que la partition courante et la règle de fusion. La première étape de la fusion consiste à sélectionner, parmi les voisins[3] de la première partie, la partie la plus fortement connectée à celle-ci. Cette nouvelle partie sera la seconde partie de la fusion. La sélection de cette seconde partie se fait de telle manière que :

$$coupe(première\_partie, seconde\_partie) = \max_{partie \in partition} coupe(première\_partie, partie) .$$

Puis les deux parties sont agrégées pour ne former plus qu'un seul ensemble de sommets dans une nouvelle partie. En accord avec les règles de fusion, un certain nombre de sommets sont sélectionnés pour être séparés de la nouvelle partie, comme décrite en sous-section 4.3.2. Les sommets qui sont le moins liés à la nouvelle partie sont sélectionnés en premier pour être incorporés à cette liste de sommets. Puis, chaque sommet de cette liste est ajouté à la partie de la partition à laquelle il est le plus fortement connecté, c'est-à-dire qui vérifie :

$$coupe(\{sommet\}, partie\_connecté) = \max_{partie \in partition} coupe(\{sommet\}, partie) .$$

Enfin, la partition est mise à jour en tenant compte du changement de partie du sommet.

4.3.8 Fission

La fission a pour but de diviser une partie de la partition en deux nouvelles parties. L'algorithme 20 décrit le processus utilisé pour réaliser la fission. La première étape de la fission consiste à sélectionner une liste de sommets qui seront séparés de la partie lors de la fission. Cette liste est construite en accord avec la règle comme décrit en sous-section 4.3.2, en sélectionnant en premier les sommets les moins fortement liés à la partie, comme dans le cas de la fusion. Puis la partie est séparée en deux parties avec l'aide d'un algorithme de bissection de graphe. Plusieurs algorithmes différents peuvent être utilisés, comme des algorithmes d'expansion de région (*supra* 2.2) ou des algorithmes multi-niveaux (*supra* 2.6). Nous avons pu constater que pour les problèmes de moins d'un millier de sommets, l'algorithme de percolation (*supra* 2.7.1) que nous avons développé était plus performant avec la

[3] les autres parties liées à cette partie par des arêtes

Algorithme 19 Fusion entre deux parties d'une partition.

Procédure FUSION(première_partie, partition, règles)
  seconde_partie ← choix d'une partie voisine de première_partie dans la partition
  nouv_partie ← agrégation de première_partie avec second_partie
  liste_des_sommets ← sommets éjectés de nouv_partie en fonction des règles
  partition ← remplace dans la partition première_partie et seconde_partie par nouv_partie
  pour tout sommet ∈ liste_des_sommets **faire**
    partie_connectée ← partie la plus fortement connectée au sommet de la partition
    partie_connectée ← ajout de sommet dans partie_connectée
    partition ← remplace partie_connectée dans la partition
  fin pour
fin Procédure

fusion-fission que les logiciels de partitionnement standards comme pMetis ou Chaco (*supra* A.1). Ces derniers ayant dans notre cas pour défaut de trouver des bissections trop bien balancées.

Après que la bissection de la partie ait été réalisée, la partition est mise à jour en remplaçant la partie initiale par les deux parties engendrées. Puis, à chaque sommet de la liste de sommets retirés de la partie initiale, est appliqué la procédure suivante :

- la partie de la partition possédant l'énergie de partie (*supra* 1.4) la plus élevée est sélectionnée, elle sera appelée max_énergie_partie ;
- l'énergie de max_énergie_partie est comparée à une énergie seuil qui est égale à l'énergie moyenne d'une partie de la partition multipliée par une constante $fission\_cha\hat{\imath}ne$, avec comme énergie moyenne d'une partie :

$$\frac{1}{k}\,\acute{e}nergie(partition)\ ;$$

  - si l'énergie de max_énergie_partie est supérieure à l'énergie seuil, alors il y a réaction en chaîne et la partie max_énergie_partie augmentée du sommet est divisée en deux parties. Ces deux nouvelles parties vont alors remplacer max_énergie_partie dans la partition ;
  - dans le cas où l'énergie de max_énergie_partie est inférieure à l'énergie seuil, le sommet est ajouté à la partie avec laquelle il est le plus lié et la partition est mise à jour, comme dans le cas de la fusion.

Le paramètre fission_chaîne permet de contrôler la fréquence d'occurrence d'une fission en chaîne.

4.3.9 Paramètres

Nous présentons dans cette sous-section la liste des paramètres utilisés par notre adaptation de la fusion-fission au partitionnement non contraint :

- le nombre de parties de la partition cherchée : k ;
- 2 règles pour la fusion, $rfusion$, et la fission, $rfission$, ainsi que $pas\_de\_r\grave{e}gle$ comme paramètre d'apprentissage de la règle ;
- l'initialisation se termine lorsqu'elle a trouvé un état de coût inférieur à $co\hat{u}t\_min$;
- pour la fonction de choix : p et q ;
- dans le mécanisme de fission : $fission\_cha\hat{\imath}ne$;
- le nombre de passes : $nombre\_de\_passes$.

Algorithme 20 Fission d'une partie d'une partition.

```
Procédure FISSION(partie, partition, règles)
    liste_des_sommets ← sommets éjectés de partie en fonction des règles
    (première_partie, seconde_partie) ← bissection de partie
    partition ← remplace partie par première_partie et seconde_partie dans la partition
    pour tout sommet ∈ liste_des_sommets faire
        max_énergie_partie ← partie de plus forte énergie de la partition
        énergie_seuil ← fission_chaîne * énergie moyenne d'une partie de partition
        si énergie de max_énergie_partie > énergie_seuil alors
            max_énergie_partie ← ajout de sommet à max_énergie_partie
            (première_partie, seconde_partie) ← bissection de max_énergie_partie
            partition ← remplace max_énergie_partie par première_partie et seconde_partie
        sinon
            partie_connectée ← partie la plus fortement connectée au sommet de la partition
            partie_connectée ← ajout de sommet dans partie_connectée
            partition ← remplace partie_connectée dans la partition
        fin si
    fin pour
fin Procédure
```

Le réglage des paramètres est en réalité moins compliqué que peut le faire croire leurs nombres. Le nombre de parties, k, est un paramètre inhérent du problème à résoudre. L'énergie maximale d'initialisation, $co\hat{u}t\_min$, dépend elle aussi du problème. Cependant, plusieurs essais peuvent être nécessaires afin de calibrer ce paramètre. Néanmoins ce paramètre peut être omis (ou initialisé à une valeur très grande), et dans ce cas, l'initialisation se terminera lorsqu'une partition de cardinal proche de k sera trouvée. Dans le cas du découpage de l'espace aérien, des solutions en moyenne presque aussi bonnes que celles où $co\hat{u}t\_min$ avait été utilisé, ont été trouvées.

Dans la pratique, il nous a semblé peu utile de changer les paramètres réglant le mécanisme de règles, à savoir $rfusion$, $rfission$, $pas\_de\_r\grave{e}gle$ et $fission\_cha\hat{i}ne$. Le mécanisme de règles ayant pour avantage de s'ajuster automatiquement, son réglage devient donc moins crucial. Nous avons réglé le paramètre $rfusion$ afin qu'en général seuls les nucléons les moins liés au nouvel atome soient éjectés, et $rfission$, afin que peu de nucléons soient éjectés des deux atomes créés. Le paramètre $pas\_de\_r\grave{e}gle$ ne dois pas être trop élevé afin de garder une certaine « inertie » de la règle afin qu'elle soit utile, mais pas trop petit non plus, afin que les modifications de la règle soient pris en compte. Enfin, le paramètre $fission\_cha\hat{i}ne$ doit être suffisamment élevé ($\geq 1,5$) pour qu'il n'y ait pas trop souvent de fission en chaîne, ce qui entraînerait trop de diversification.

Les paramètres p et q doivent résulter d'un équilibre avec le paramètre $nombre\_de\_passes$. Cependant, il est préférable de fixer rapidement ces deux premiers paramètres afin de ne faire varier que $nombre\_de\_passes$. Nous avons remarqué que ce dernier devait augmenter avec la taille du problème et la valeur de k.

Ainsi, les paramètres à fixer rapidement et ne nécessitant pas de nombreux réglages sont les suivants : $rfusion$, $rfission$, $pas\_de\_r\grave{e}gle$, $fission\_cha\hat{i}ne$, p et q. Les paramètres de réglages plus fin de l'algorithme sont donc : k, $nombre\_de\_passes$, $co\hat{u}t\_min$. Comme nous pouvons le constater, il ne reste assez rapidement que deux paramètres à régler pour utiliser cet algorithme sur différentes instances du problème.

4.4 Application au partitionnement contraint

4.4.1 Présentation

Nous avons cherché à utiliser la méthode de fusion-fission dans le cadre du partitionnement contraint afin d'évaluer cette méthode sur des bancs de tests classiques. Les principaux bancs de tests dans le cas non contraint sont ceux de la segmentation d'image. Or, il est assez difficile d'effectuer des comparaisons avec des segmentations d'images déjà trouvées, car une « bonne » segmentation s'évalue souvent plus visuellement que grâce à une fonction de coût. D'un autre côté, le partitionnement contraint dispose d'un nombre très important de graphes de tests et de logiciels de partitionnement disponibles, mais aussi, grâce au site de Chris Walshaw (*infra* annexe A.4), de nombreux graphes de tests auxquels sont associés des partitions.

Avant de chercher à réadapter la méthode de fusion-fission au partitionnement contraint, nous avons modifié l'adaptation qui avait été faite pour le partitionnement contraint, présenté dans la section précédente, afin de prendre en compte les spécificités du cas contraint. Cependant, les résultats obtenus avec l'algorithme ainsi modifié étaient très insatisfaisants. En effet, les partitions trouvées ne respectaient presque jamais la balance de partitionnement et les quelques fois où elles y arrivaient, leurs coûts de coupe étaient bien supérieurs à ceux trouvés par les méthodes multi-niveaux (*supra* 2.6).

Ce constat ne doit pas étonner le lecteur. En effet, si, en premier abord, le partitionnement contraint semble très proche du cas non contraint, les buts recherchés sont ordonnés différemment (*supra* 1.7.1), ce qui différencie sensiblement les problèmes. L'absence d'article dans la littérature présentant les différences entre ces deux problèmes est une lacune qui peut faire croire qu'ils sont identiques.

L'adaptation directe des métaheuristiques au partitionnement contraint donne souvent des résultats médiocres en comparaison de ceux des méthodes multi-niveaux. Par contre, les méthodes hybrides, métaheuristiques et multi-niveaux, donnent de très bon résultats, en terme de qualité de coupe des partitions trouvées, malgré un temps d'exécution plus long [SOP04b, KOR04]. Après ces constatations, nous avons pensé introduire une méthode multi-niveaux dans le processus de fusion-fission puisqu'il paraissait que c'était une méthode indispensable actuellement au partitionnement de graphe, comme les méthodes d'affinage de type Kernighan-Lin le sont aussi devenus il y a une décennie. Notre adaptation de la fusion-fission au cas contraint comporte donc une méthode multi-niveaux et une méthode d'affinage de type Kernighan-Lin.

Après avoir appliqué notre méthode de fusion-fission au découpage de l'espace aérien (*infra* 5), nous avons voulu la comparer à d'autres méthodes de partitionnement. Or, la plus grande partie des méthodes de partitionnement sont uniquement adaptées au cas contraint (*supra* A). Nous avons donc voulu adapter notre méthode au partitionnement de graphe contraint. Pour bâtir un nouvel algorithme de fusion-fission adapté à ce nouveau problème, nous nous sommes réappropriés les principes de la fusion-fission énoncés en section 4.2 et les notions de physique nucléaire présentés en section 4.1.

Dans la perspective d'avoir un nombre de parties toujours changeant, le fait de respecter une balance de partitionnement est une contrainte très forte. Il a fallu trouver un mécanisme permettant d'inclure cette contrainte dans le mécanisme de fusion-fission. Dans l'adaptation de la fusion-fission au partitionnement non contraint, deux phénomènes font que la partition finale ne respecte plus la balance de partitionnement :

- dans le cas de la fission, la bissection d'une partie de la partition déséquilibre celle-ci ;
- dans le cas de la fusion, l'agrégation de deux parties de la partition la déséquilibre.

Afin de pallier à ces problèmes, nous avons choisi une technique simple qui consiste à appliquer la fission simultanément à toutes les parties de la partition, puis à utiliser la fusion sur toutes les parties ainsi créées. Cette technique simplifie le mécanisme utilisé dans le cas non contraint. En effet, il n'est plus utile de choisir une partie aléatoirement à chaque itération, ni de déterminer si cette partie va subir une fusion ou une fission. Toutes les parties de la partition sont affectées simultanément, dans le cas de la fission comme dans celui de la fusion. Cependant, pour garder l'esprit initial de la méthode de fusion-fission consistant à avoir un nombre de parties fluctuant, il a fallu introduire un mécanisme permettant de choisir, à chaque étape, quel nombre de parties la nouvelle partition devra avoir.

Le mécanisme de règle a lui aussi évolué. Ce dernier visait à éjecter les sommets peu liés à la partie. Bien que ce mécanisme aurait pu être réutilisé tel quel, les algorithmes d'affinage habituels (de type Kernighan-Lin *supra* 2) effectuent déjà ce travail. Dans un souci de simplification de la méthode, nous avons choisi de ne pas utiliser le mécanisme de règle en plus d'un algorithme d'affinage.

4.4.2 Algorithme

Dans la sous-section précédente, nous avons vu les grandes lignes de l'adaptation de la fusion-fission au partitionnement contraint. Cette sous-section va présenter cette adaptation dans le détail.

L'algorithme 21 présente l'adaptation de la méthode de fusion-fission au partitionnement contraint. Cet algorithme est composé d'une partie d'initialisation et d'une boucle itérative appelée boucle sur le nombre de parties :

- l'initialisation de l'algorithme se fait en deux étapes :
  - dans un premier temps est créée une séquence d'entiers qui serviront dans la boucle sur le nombre de parties, à donner le nombre de parties de la partition à créer. En effet, le nombre de parties des partitions trouvées par la boucle sur le nombre de parties évoluant à chaque itération, il faut connaître au début de chacune de ces itérations quelle sera le nombre de parties de la nouvelle partition à créer. Nous avons délibérément choisi de créer la liste de ces nombre de parties au début de l'algorithme, bien que d'autres solutions soient envisageables. Un nombre n d'itérations de la boucle sur le nombre de parties est donné en argument de l'algorithme, et celui-ci s'arrêtera lorsque ce nombre sera atteint. La création de cette séquence de nombre de parties est l'objet de la sous-section 4.4.4 ;
  - dans un second temps, une partition initiale en k parties est créée. Plusieurs choix quant à la méthode de partitionnement sont possibles. Cependant, comme nous l'avons indiqué dans la sous-section précédente, dans un souci de performance, nous avons tenu à utiliser une méthode multi-niveaux. Cette méthode étant utilisée dans la boucle sur le nombre de parties, il est facile de l'utiliser pour créer la partition initiale. Comme cette solution est rapide et performante, c'est celle qui a été choisie. Comme nous pourrons le voir plus loin (*infra* 4.4.3), c'est l'algorithme multi-niveaux pMetis qui a été choisi comme outil de partitionnement. La partition du graphe initiale en k parties trouvée par le l'outil de partitionnement ($partitionneur$, pMetis) est enregistrée dans P^k. P^k permet de sauvegarder la partition la plus performante trouvée pendant la boucle sur le nombre de parties alors que P est la partition courante dans cette boucle ;
- succède à l'initialisation la boucle sur le nombre de parties. Cette boucle effectue n itérations, chacune se décomposant en six étapes :
  - la première étape d'une itération de la boucle consiste à déterminer le nombre de parties que devra avoir la nouvelle partition qui sera créée par cette itération. Ce nombre de parties est k_i à l'itération i et a été déterminé par la séquence des nombres de parties. La partition courante P possède quant à elle k_{i-1} parties ;
  - la seconde étape est celle de la fission. Comme nous l'avions dit dans la sous-section précédente (*infra* 4.4.1), l'étape de fission consiste à diviser chaque partie de la partition courante P en plusieurs parties. Afin de respecter autant que possible la balance de partitionnement et

dans la perspective où la fusion va créer une partition à partir des parties créées par la fission, chaque partie de la partition P va être divisée en k_i nouvelles parties. En effet, après la fission, l'ensemble des parties de la partition P contiendra $k_{i-1} * k_i$ parties, ce qui permettra, lors de l'étape de fusion, de créer plus facilement une partition de k_i parties. Pour ce faire, chacune des k_{i-1} parties de P est segmentée en k_i parties par le l'outil de partitionnement ;
- la troisième étape consiste à préparer l'étape de fusion. Pour cela, un graphe temporaire G' est créé tel que chaque sommet du graphe soit une des $k_{i-1} * k_i$ parties de l'ensemble des parties de P . Les arêtes de ce graphe ont pour poids la somme du poids des arêtes reliant deux parties et les sommets ont pour poids la somme du poids des sommets constituant chaque partie ;
- la quatrième étape est celle de fusion. Elle a pour but de créer une partition en k_i parties à partir du graphe G' qui vient d'être construit. Cette partition est créée à l'aide de l'outil de partitionnement. Elle doit être de coût de coupe le plus bas possible tout en respectant, avec une certaine tolérance, la balance de partitionnement. En effet, nous avons constaté qu'avec l'étape d'affinage qui suivait, le coût de coupe était légèrement plus important que la balance à cette étape. Une fois la partition en k_i parties créée à partir du graphe G', il faut projeter cette partition sur le graphe initial G. Cette projection est triviale, puisque l'on connaît l'appartenance de chaque sommet de S à une des parties de l'ensemble des parties de P. Enfin, la partition courante P peut être mise à jour ;
- la cinquième étape est celle d'affinage de la partition courante P. L'affinage remplace l'étape d'éjection des nucléons de l'atome présente dans le modèle de fusion-fission. Cependant, l'affinage fonctionne comme le système de règle utilisé dans le cas non contraint dans la mesure où il permet aux sommets peu liés aux parties de s'en séparer. L'un des avantages de l'affinage sur le système de règle que nous avions modélisé est qu'il est plus classique, l'inconvénient est qu'il ne possède aucun mécanisme d'apprentissage. Le choix de l'outil d'affinage est présenté en sous-section 4.4.5. Il nous est apparu après de nombreux essais que l'étape d'affinage était plus utile pour trouver la meilleure partition P^k que pour être utilisée comme partition courante. En effet, ne pas utiliser la partition affinée comme partition courante permettait à l'algorithme d'explorer plus de minima locaux que dans le cas contraire. Cette constatation, à première vue contre intuitive, nous a poussés à utiliser la partition non affinée au détriment de celle affinée lors de l'itération suivante ;
- la sixième et dernière étape permet de mettre à jour la partition la plus performante P^k. Le remplacement de P^k par P' n'a lieu que si $card(P') = k$ et si la partition affinée à un coût de coupe moindre que P^k tout en respectant la balance de partitionnement imposée. Cependant, les meilleures partitions possédant un nombre de parties différent de k, peuvent aussi être sauvegardées. Cela nous a permis de confirmer le fait que de bonnes partitions en k' parties permettaient de trouver de bonnes partitions en k parties.

4.4.3 Choix de l'algorithme multi-niveaux

Le choix de la méthode multi-niveaux à utiliser ne fut pas simple. Nous avons testé plusieurs possibilités :
- nous avons écrit un algorithme multi-niveaux de bissection récursif (*supra* 1.7.3) de graphe. Cependant, nous nous sommes rendus compte que notre algorithme était beaucoup moins optimisé que les logiciels existants tant pour le temps de calcul nécessaire que pour la performance obtenue. Nous avons donc arrêté rapidement cet essai pour adapter une librairie classique ;
- nous avons utilisé l'algorithme multi-niveaux de la librairie Chaco. Cette solution ne nous a

Algorithme 21 Adaptation de la fusion-fission au partitionnement contraint.

Procédure FUSIONFISSIONCONTRAINT($G = (S, A)$, k, n, $partitionneur$)
  % Début de l'étape 1 de l'initialisation
  Créer une séquence aléatoire de n entiers suivant une loi binomiale centrée en k : $\{k_1, \ldots, k_n\}$
  $k_0 \leftarrow k$
  % Début de l'étape 2 de l'initialisation
  $P^k \leftarrow partitionneur(S, k)$
  $P \leftarrow P^k = \{P_1, \ldots, P_k\}$
  pour tout $k_i \in \{k_1, \ldots, k_n\}$ **faire** % Boucle sur le nombre de parties, étape 1
    pour $j = 1$ **à** k_{i-1} **faire** % Étape 2
      Partitionner la partie $P_j \in P$ en k_i nouvelles parties avec $partitionneur(P_j, k_i)$
      $ensemble\_parties$ ← ajouter les k_i nouvelles parties crées
    fin pour % Début de l'étape 3
    Créer un graphe $G' = (S', A')$ où chaque sommet est une des parties de $ensemble\_parties$
    $P = \{P_1, \ldots, P_{k_i}\} \leftarrow partitionneur(S', k_i)$ % Étape 4
    P' ← affinage de P % Étape 5
    si $k_i = k$ **et** $coupe(P') < coupe(P^k)$ **alors** % Étape 6
      $P^k \leftarrow P'$
    fin si
  fin pour
  retourner P^k
fin Procédure

pas satisfaits, car l'algorithme était trop strict au niveau de la balance de partitionnement de partitions qu'il cherchait. Or, lors du partitionnement du graphe G', il est préférable d'obtenir une partition possédant un coût de coupe petit mais une balance de partitionnement élevée que l'inverse, car la balance peut plus facilement être récupérée par les algorithmes d'affinage que le coût de coupe ;
– les algorithmes pMetis et kMetis de la librairie Metis ont été adaptés. L'algorithme pMetis produisait des partitions ayant une balance de partitionnement trop déséquilibrée. Ainsi, les résultats obtenus avec pMetis n'étaient pas bons car les algorithmes d'affinage n'arrivaient pas à faire suffisamment descendre la balance de partitionnement des partitions trouvées par pMetis. Cependant, l'algorithme pMetis a très bien fonctionné. Il permet de trouver des partitions performantes en terme de coût de coupe, avec une balance de partitionnement pouvant être lâche sans cependant être toujours trop grande. C'est cet algorithme qui est utilisé comme méthode multi-niveaux pour cette adaptation de la fusion-fission au cas contraint ;
– nous aurions pu utiliser l'algorithme multi-niveaux de la librairie Jostle, l'une des dernières librairies facile à utiliser et dont le code source est disponible. Malheureusement, par manque de temps nous ne l'avons pas fait, ce serait pourtant à essayer car c'est la librairie la plus flexible que nous connaissons et l'on pourrait donc l'adapter plus finement que les autres.

4.4.4 Création de la séquence des nombres de parties

La variation des nombres de parties des partitions pendant l'itération de l'algorithme est inspirée de la variation du nombre d'atomes lors des processus de fusion et de fission nucléaire. À nombre de nucléons constant, la variation du nombre d'atomes entraîne une variation du type de ces atomes, le but étant de créer principalement des atomes de fer. Pour autant, si le nombre de nucléons permet d'obtenir théoriquement k atomes de fer, un état possédant k atomes peut ne posséder aucun atome de fer. Ce n'est que la réorganisation successive des nucléons entre eux qui permet d'obtenir k atomes de fer.

Pour créer la séquence des nombres de parties, nous nous sommes inspirés de la courbe d'Aston (*supra* figure 4.2). Localement, autour de l'atome de fer, la courbe a la forme d'une petite colline aplatie, ce qui veut dire que le atomes proches du fer possèdent aussi une énergie de liaison forte. Afin d'imiter la courbe d'Aston et par souci de simplicité, pour choisir les nombres de parties, nous avons utilisé une distribution binomiale centrée en k (le nombre de parties cherchées par l'algorithme) et de variance élevée afin de garder sa forme aplatie.

La loi de probabilité binomiale s'écrit :

$$p_b(k_b) = C_{n_b}^{k_b}\, p_b^{k_b}\, (1 - p_b)^{n_b - k_b} \; .$$

Son espérance est $n_b p_b$ et sa variance $n_b p_b(1 - p_b)$. La loi binomiale possède une propriété que nous allons utiliser :

$$\sum_{k=0}^{n_b} p_b(k) = 1 \; .$$

Dans le cadre des nombres de parties qui nous intéressent, à savoir les puissances de 2 allant de 2 à 64, nous avons choisi $n_b = k + 16$ et $p_b = \frac{k}{n_b}$ afin que la courbe soit suffisamment plate et les nombres de parties suffisamment différents. Le nombre n_b correspond au nombre maximal de parties qu'une partition pourra posséder pendant l'optimisation.

La première étape de la création de la séquence des nombres de parties est de calculer pour chaque nombre de parties potentiel sa probabilité d'apparition. Les nombres de parties potentiels sont ceux entre $n_b - k$ et $n_b + k$. Or, comme la somme des probabilités d'une loi binomiale vaut 1, pour obtenir n itérations de la boucle sur le nombre d'itérations, chacune de ces probabilités est multipliée par n. Ces nombres sont ensuite tronqués à leur valeur entière inférieure. Nous obtenons alors le nombre d'apparitions de chaque nombre de parties $k' \in \{n_b - k, \ldots, n_b + k\}$ dans la boucle : $\lceil n * p_b(k') \rceil$. Cependant, à cause de la troncature de ces valeurs, leur somme est strictement inférieure à n. La différence entre leur somme et n est ajoutée au nombre d'occurrences du nombre de parties k. Ainsi, chaque nombre d'occurrences $no(k')$ du nombre de parties $k' \in \{n_b - k, \ldots, n_b + k\}$ vaut :

$$\text{Si } k' \neq k, \text{ alors} \quad no(k') = \lceil n * p_b(k') \rceil \; ,$$
$$\text{sinon,} \quad no(k) = n - \sum_{k'=n_b-k, k'\neq k}^{n_b+k} \lceil n * p_b(k') \rceil \; .$$

La seconde étape consiste à créer une séquence de nombres de parties $k' \in \{n_b - k, \ldots, n_b + k\}$ tel que chacun de ces nombres de parties k', apparaisse exactement $no(k')$ fois dans la séquence. La séquence subit alors une permutation aléatoire pour mélanger les nombres de parties. Cette étape achève la création de la séquence des nombres de parties.

4.4.5 Choix de l'algorithme d'affinage

Il existe un grand nombre d'algorithmes d'affinage pour le partitionnement de graphe (*supra* 2.5). En choisir un en particulier s'est donc révélé complexe. Pour en choisir un pertinent, nous avons établi quelques critères de sélection :

- la partie d'affinage dans le processus d'optimisation multi-niveaux occupe la plus grande partie du temps de l'optimisation. Fort de cette expérience, et afin de ralentir le moins possible l'algorithme de fusion-fission, l'algorithme d'affinage choisi doit être rapide ;
- l'algorithme d'affinage doit pouvoir s'adapter à plusieurs balances de partitionnement différentes ;
- bien entendu, comme le but est de trouver une partition de coût de coupe minimal, l'algorithme d'affinage doit être efficace.

Ces trois critères nous ont poussés à choisir un algorithme d'affinage de type Kernighan-Lin utilisant l'implémentation de Fidducia-Mattheyses pour plus de rapidité. L'un des algorithmes d'affinage de ce type les plus rapides est l'algorithme *Global Kernighan-Lin refinement* (GKLR, *supra* 2.5.3). C'est lui que nous avons choisi comme méthode d'affinage.

L'étape d'affinage est la dernière étape avant la mise à jour de la partition la plus performante P^k. De plus, cette étape n'influence pas la partition courante de la boucle sur le nombre de parties. Dans le but de trouver des partitions pour les bancs de tests, plutôt que de lancer plusieurs fois l'algorithme pour des balances de partitionnement différentes, nous avons choisi d'appliquer plusieurs fois l'algorithme d'affinage à la même partition avec des balances de partitionnement différentes. Il faut dans ce cas utiliser autant de partitions les plus performantes trouvées qu'il y a de balances imposées. Et la partition la plus performante est mise à jour en fonction de la balance de partitionnement de la partition affinée.

4.4.6 Améliorations apportées à l'algorithme

De nombreuses modifications ont été apportées à l'algorithme de fusion-fission pour le partitionnement contraint. Par rapport à la première version de l'algorithme, certaines modifications comme l'utilisation d'une loi de probabilité binomiale ou l'affinage des partitions trouvées ont déjà été présentées ci-dessus en même temps que l'algorithme. Cependant, d'autres modifications intéressantes ont été apportées depuis :

- diminution du nombre d'affinages ;
- modification de l'algorithme d'affinage ;
- utilisation d'un algorithme de répartition de charge (*load balancing* en anglais) ;
- adaptation de la séquence des nombres de parties au cas où le carré du nombre maximal de parties d'une partition est de la même taille que le nombre de sommets ;
- modification de la séquence des nombres de parties.

Diminution du nombre d'affinages

Nous sommes restés très longtemps attachés au principe de minimiser le coût de coupe de toutes les partitions trouvées durant la boucle sur le nombre de parties. Cette persistance est en partie due à l'importance dans le processus de fusion-fission de l'éjection de certain nucléons, qui est, dans notre adaptation, associée à la phase d'affinage. Cependant, seule la partition non affinée est utilisée à l'itération suivante par cette même boucle. Ainsi, si la partition cherchée est, comme dans la majorité des cas, uniquement une partition en k parties, alors il est inutile d'affiner les partitions dont le nombre de parties est différent de k. Adopter une telle stratégie ne change en rien la performance de l'algorithme, mais l'accélère avantageusement. En effet, comme nous l'avons vu, la phase d'affinage est l'une des plus consommatrices en temps de calcul.

Utilisation de l'algorithme d'affinage de Walshaw-Cross

L'algorithme d'affinage GKLR utilisé par l'adaptation de la fusion-fission s'est révélé très performant lorsque le nombre de parties de la partition était peu élevé ($k = 2$ ou 4). Cependant, pour des nombres de parties plus élevés ($k = 16$ ou 32), l'algorithme d'affinage GKLR avait beaucoup de difficulté à trouver des partitions respectant la balance de partitionnement. Il faut ajouter à la décharge de l'algorithme GKLR qu'il n'est pas conçu pour faire de la répartition de charge, et que la balance des partitions courantes dans la boucle sur le nombre de parties est en moyenne légèrement plus élevée pour un nombre de parties grand que pour un nombre de parties petit.

Nous avons donc cherché à pallier les insuffisances de l'algorithme GKLR par l'utilisation d'un autre algorithme d'affinage. Celui-ci, en plus de répondre aux exigences sur l'algorithme d'affinage présenté en sous-section 4.4.5, doit pouvoir dans tous les cas retourner une partition respectant la balance de partitionnement, quitte à ce que son coût de coupe diminue fortement. Or l'algorithme d'affinage de Walshaw-Cross (*supra* 2.5.4) utilise une méthode de répartition de charge afin d'équilibrer la partition. De plus, son algorithme d'affinage est rapide puisque qu'il est basé sur l'implémentation de Fiduccia-Mattheyses. Nous avons créé une version de l'algorithme de fusion-fission qui utilisait la méthode d'affinage de Walshaw-Cross à la place de l'algorithme GKLR.

Utilisation d'un algorithme de répartition de charge en complément de l'algorithme GKLR

L'utilisation de l'algorithme d'affinage de Walshaw-Cross présenté ci-dessus par la fusion-fission a donné de bons résultats. Cependant cette technique d'affinage est assez compliquée à mettre en œuvre, car elle utilise plusieurs méthodes différentes. Nous avons voulu créer une méthode plus simple et uniquement basée sur le principe de gain de Kernighan-Lin. Nous espérions aussi trouver une méthode un peu plus rapide. Or, puisque nous disposions d'un bon algorithme d'affinage, à savoir l'algorithme GKLR, il nous suffisait de trouver un algorithme de répartition de charge. Nous avons donc créé l'algorithme de répartition de charge qui est présenté en section 2.7.2.

Comme nous l'avons dit, cet algorithme de répartition de charge est utilisé en complément de l'algorithme GKLR. Cependant, durant les itérations de la boucle sur le nombre de parties, certaines partitions courantes respectent déjà la balance de partitionnement. Dans ce cas, il est inutile de leur appliquer l'algorithme de répartition de charge avant l'algorithme GKLR. D'autre part, dans certains cas, l'algorithme de répartition de charge trouve une partition de coût de coupe inférieure à celle de l'algorithme GKLR ou encore l'algorithme GKLR trouve une partition ne respectant pas la balance de partitionnement. Dans ces deux cas, la partition la plus performante P^k est mise à jour par comparaison avec la partition trouvée par l'algorithme de répartition de charge.

Adaptation de la séquence des nombres de parties au cas où le carré du nombre maximal de parties d'une partition est de la même taille que le nombre de sommets

Nous avons vu dans la sous-section 4.4.4 comment la séquence des nombres de parties Seq était créée. Notre adaptation de la fusion-fission commence par partitionner chaque partie de la partition courante avant d'agréger les parties ainsi formées entre elles pour créer une nouvelle partition. Supposons que la partition courante est $P = \{P_1, \ldots, P_k\}$ et que le nombre de parties de la future partition soit k'. Chaque partie $P_i \in P$ devra être partitionnée en k' parties. Cependant, cela pose un problème si $k' > card(P_i)$, puisqu'il n'y aura pas assez de sommets à partitionner.

Pour contrer cet inconvénient, lorsque le nombre de parties k, le nombre de parties maximal n_b et le nombre de sommets $nb\_sommets$ vérifient :

$$(k+16)^2 = n_b^2 > 90\% * nb\_sommets \ ,$$

alors entre chaque partition respectant la séquence Seq, est créée une partition dont le nombre de parties suit une autre séquence que Seq. Cette nouvelle séquence Seq' est toujours basée sur une loi de probabilité binomiale, mais celle-ci est centrée en un nombre de parties $kpetit$ tel que :

$$kpetit = \ 90\% * \frac{card(S)}{k} \ .$$

Pour le reste, cette seconde séquence Seq' est construite de la même façon que la séquence S.

Le choix du pourcentage 90% est empirique. Dans nos essais, il correspond à la limite haute du domaine des valeurs utilisables. En utilisant le pourcentage 70%, nous n'avons jamais rencontré le cas où $k' > card(P_i)$, alors que ce cas apparaît de temps en temps avec 90%. Cependant, notre algorithme est assez robuste pour supporter ce problème. Dans ce cas, le graphe engendré comporte moins de sommets, mais cela n'handicape pas la création d'une nouvelle partition en k' parties. Nous conseillons donc de choisir ce pourcentage entre 70% et 90%, car au-delà, ce problème apparaît trop fréquemment et nuit à la performance de l'algorithme, et en dessous, cette adaptation est inutile.

Modification de la séquence des nombres de parties

Plutôt que de choisir comme critère d'arrêt d'une algorithme itératif son nombre d'itérations, il peut être intéressant de le limiter dans son temps d'exécution. Dans ce cas, la création d'une séquence des nombres de parties dont la longueur valait le nombre d'itérations, n'est plus adaptée. Nous avons donc utilisé une astuce pour permettre à l'algorithme de s'arrêter après un temps d'exécution prédéfini. Une séquence des nombres de parties de longueur égale au nombre de sommets du graphe est créée. Puis à chaque itération de la boucle sur le nombre de parties, un nombre de parties est choisi aléatoirement dans cette séquence.

D'autre part, nous avons constaté que, lorsque k était grand, le nombre de parties k était peu présent dans la séquence, ce qui entraînait de moins bonnes performances de l'algorithme que dans le cas où k était petit. Pour régler ce problème, bien qu'elle fausse un peu l'utilisation de la distribution binomiale dans le cas de la probabilité d'occurrence de k, la solution suivante fut adoptée : lorsqu'une boucle sur le nombre de parties a itéré $nb\_iter\_sans\_k$ fois sans créer une partition de k parties, alors la prochaine itération devra créer une partition de k parties.

4.4.7 Autres améliorations possibles de l'algorithme

Nous présentons dans cette sous-section quelques voies simples permettant d'améliorer la rapidité et/ou l'efficacité de notre adaptation de la fusion-fission. Nous n'avons pas encore eu le temps de les tester, c'est pourquoi nous les présentons séparément.

Ne créer que des k-partitions

La variation du nombre de parties des partitions pendant l'algorithme de fusion-fission a pour origine la variation du nombre d'atomes lors des processus de fusion et de fission nucléaire. Cependant, nous avons pu observer qu'une intensification du nombre d'occurrences des partitions en k parties dans la séquence des nombres de parties permettait de trouver plus rapidement de bon résultats. Nous pensons donc qu'en vue de créer un algorithme plus rapide, il pourrait être intéressant de ne créer que des k-partitions à partir de fissions en des nombres d'éléments différents.

Ainsi, une suite u de nombres pourrait être créée telle que la fission de toutes les parties d'une k-partition en u_i éléments produise au total entre 40 et 200 éléments. En effet, un graphe contenant entre 40 et 200 sommets est très facilement et rapidement partitionnable. La suite u devrait être courte pour que l'algorithme de fusion-fission soit rapide.

Une implémentation parallèle simple

Une implémentation en parallèle très simple de l'algorithme de fusion-fission permettrait d'accélérer considérablement sa vitesse. Comme nous l'avons vu dans la sous-section sur le choix de l'algorithme d'affinage (*supra* 4.4.5), l'algorithme d'affinage monopolise un temps considérable de l'exécution du programme. Or la partition affinée résultante n'est pas utilisée à l'itération suivante de la boucle sur le nombre de parties. Il serait donc facile de confier les tâches d'affinage et de mise à jour de la meilleure partition trouvée à un autre processeur. Avec la popularisation des noyaux double-cœur, cette amélioration ne va monopoliser qu'un ordinateur de bureau comme précédemment, mais en apportant plus de rapidité.

4.5 Perspectives futures

Aucune adaptation de la méthode de fusion-fission à d'autres problèmes d'optimisation combinatoire n'a été réalisée. Nous aurions pu le faire, mais nous avons préféré adapter la fusion-fission au partitionnement contraint. En effet, après avoir utilisé des outils de partitionnement classiques, et joué avec leurs paramètres pour obtenir des solutions performantes au problème du découpage de l'espace aérien, nous voulions adapter la méthode de fusion-fission au problème pour lequel ces outils avaient été conçus. Une fois l'adaptation réalisée, nous voulions pouvoir comparer de manière objective notre méthode avec ces outils classiques. Mais c'est aussi parce qu'une adaptation à un autre problème que celui du partitionnement de graphe nous éloignait trop de notre sujet de thèse que nous ne l'avons pas faite.

Cependant, il serait intéressant de généraliser la méthode de fusion-fission pour en faire une métaheuristique. Dans certain de nos papiers [BIC04b, BIC06a] nous l'avons déjà présenté comme une métaheuristique, cependant, nous avons conscience que son adaptation à d'autres problèmes que celui du partitionnement de graphe reste difficile.

La méthode de fusion-fission a été conçue pour résoudre un problème de partitionnement de graphe. Cependant, elle résout ce problème différemment de l'approche classique. En effet, elle permet de trouver des partitions de cardinaux différents, quand les méthodes classiques ne s'intéressent uniquement qu'à un nombre de parties. Nous avons tout de suite été intéressés par la possibilité de trouver en une exécution des découpages de l'espace aérien européen en un nombre de parties différentes, car ce paramètre n'était pas précis. Notre méthode a donc été construite autour de ce problème : trouver plusieurs partitions de cardinaux différents en une exécution. C'est à ce moment là que nous est venue l'idée de l'analogie avec les réactions nucléaires de fusion et de fission. En effet, l'analogie avec la création d'une partition par étapes successives était directe, et le mécanisme de fusion-fission occasionnait un changement inhérent du cardinal de la partition cherchée.

L'adaptation de la fusion-fission à d'autres problèmes combinatoires fait partie de nos projets futurs de recherche. Mais pour s'y attaquer, il nous faut commencer à répondre à quelques questions. L'une d'elles est de savoir quelle est la différence entre la fusion-fission et les autres métaheuristiques.

La méthode multi-niveaux a été récemment adaptée à d'autres problèmes d'optimisation combinatoire [WAL04], pourtant cette adaptation semble ne pas avoir été suivie. La méthode de fusion-fission serait-elle trop liée au partitionnement de graphe pour être vraiment utilisée à la résolution d'autres problèmes ?

Nous pouvons cependant commencer à proposer quelques pistes pour une future adaptation. Il nous semble qu'une des originalités de cette méthode est qu'elle trouve des partitions performantes pour plusieurs nombres de parties différents. Or, dans le problème général du partitionnement de graphe (*supra* 1.3), le cardinal d'une partition fait partie des contraintes du problème, c'est même à ce niveau la seule contrainte du problème. Partant de ce constat, nous pouvons dire qu'une adaptation de la fusion-fission devra sûrement dans son processus de résolution du problème faire varier de manière locale la ou les contraintes du problème. D'autre part, la méthode procède par fusion des caractéristiques internes de l'individu optimisé, et par division et réévaluation de ces caractéristiques.

Cependant, ces pistes sont pour l'instant trop vagues pour permettre une généralisation précise de la méthode.

4.6 Conclusion

Nous avons présenté dans ce chapitre une nouvelle méthode d'optimisation combinatoire pour le partitionnement de graphe, la méthode de fusion-fission. Celle-ci est la principale contribution de cette thèse. Ce chapitre a commencé par introduire quelques notions de physique nucléaire sur lesquelles la méthode de fusion-fission repose. Puis les bases de la fusion-fission ont été décrites. Ces bases reprennent les mécanismes physiques fondamentaux de la fusion et de la fission entre atomes. L'idée de base de la méthode est de simuler des fusions et des fissions entre nucléons, ceux-ci étant les paramètres élémentaires du problème d'optimisation. Après cette description, ont été présentées les deux applications de la fusion-fission aux problèmes du partitionnement de graphe non contraint, puis contraint. Ces présentations sont largement détaillées afin de pouvoir en saisir toutes les nuances. Enfin, ont été proposées des indications sur les perspectives futures la fusion-fission, la plus ambitieuse étant d'en faire une métaheuristique à part entière, c'est-à-dire applicable à tout problème d'optimisation.

Comme nous le verrons, la méthode de fusion-fission a été utilisée pour résoudre les problèmes du découpage de l'espace aérien, de la segmentation d'image et de la classification de documents. Elle a aussi permis de trouver de nouvelles solutions sur les bancs de tests classiques du partitionnement de graphe contraint.

Deuxième partie

Découpage de l’espace aérien et autres applications

Chapitre 5

Application des méthodes de partitionnement de graphe au découpage de l'espace aérien européen

L'étude du découpage de l'espace aérien européen est le sujet à la base de cette thèse. Si celle-ci a évolué progressivement vers des aspects plus théoriques du partitionnement de graphe, le découpage de l'espace aérien reste l'application principale des méthodes qu'elle étudie.

Dans une première section nous allons présenter au lecteur néophyte les notions relatives à la structure de l'espace aérien dont nous nous servirons par la suite (*infra* 5.1). Puis dans une seconde section nous présenterons le problème posé par le redécoupage de l'espace aérien actuel (*infra* 5.2). La troisième section aura pour objet la modélisation du problème et l'adaptation des algorithmes de partitionnement à ce problème (*infra* 5.3). La quatrième partie présentera les résultats obtenus (*infra* 5.4). Enfin, la dernière partie fera la synthèse du travail présenté et proposera de nouvelles directions afin de rendre les résultats de cette étude plus opérationnels (*supra* 5.5).

5.1 Structure de l'espace aérien

Le but de cette section est de présenter au lecteur néophyte l'organisation de l'espace aérien tel qu'il existe en Europe, mais aussi dans le reste du monde.

La façon la plus intuitive pour présenter cette organisation est de commencer par décrire les différents types de trajets que peuvent emprunter les avions. Une fois que cette description sera achevée, la présentation de la structure de contrôle permettant aux avions de naviguer en sécurité, sera plus compréhensible.

5.1.1 Le réseau de routes aériennes

Il existe deux modes de vol différents lorsque l'on souhaite effectuer un trajet en avion : le vol à vue, en anglais *Visual Flight Rules* (VFR), et le vol aux instruments, en anglais *Instruments Flight Rules* (IFR). Le vol aux instruments nécessite une qualification supplémentaire par rapport au vol à vue. De plus, pour qu'un aéronef puisse voler aux instruments, il doit être doté d'équipements spécifiques supplémentaires (principalement de radionavigation). La quasi totalité des vols commerciaux volent aux instruments.

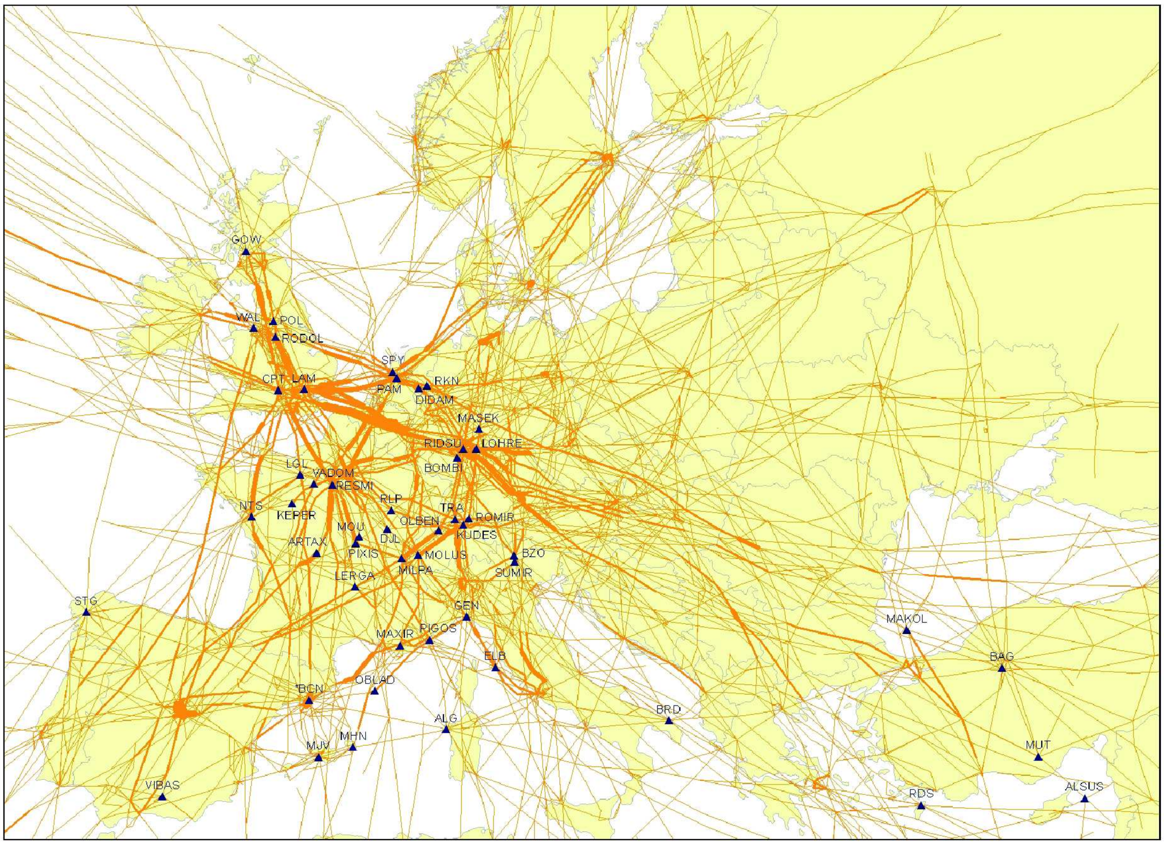


FIG. 5.1 – Réseau des principales routes aériennes en Europe.

Un aéronef volant aux instruments voulant aller d'un aérodrome à un autre suit un parcours segmenté dont les points d'intersection correspondent à des balises de navigation. Ce parcours segmenté suivi par l'aéronef est appelé *route aérienne*. Les balises de navigation sont aussi appelées points de report ou *waypoints* en anglais. Ces points de report sont soit des balises de radionavigation physiquement implantées au sol, soit des points de report « fictifs ». Le pilote se repére par rapport à ces balises grâce aux moyens de radionavigation embarqués, chaque balise au sol émettant un signal sur une fréquence lui étant propre. Historiquement, les points de report correspondaient à des balises de radionavigation physiquement implantées au sol. Pour des raisons de visibilité, ces balises étaient souvent placées en fonction du relief, ce qui donnait aux routes un aspect parfois tortueux. L'amélioration des moyens de communication embarqués a permis le développement d'un second type de points de report. Ceux-ci sont fictifs dans la mesure où ils ne correspondent à aucun équipement physique au sol.

Sur la figure 5.1 est dessiné le réseau des grandes routes aériennes en Europe. Sur cette figure, chaque triangle représente l'une des 50 balises les plus contraignantes d'Europe, c'est-à-dire qui dévie le plus grand nombre d'avions par rapport à une trajectoire directe. L'épaisseur des traits représente la densité du trafic entre les balises, plus ce trait est épais, et plus le nombre d'aéronefs passant entre ces balises est élevé.

Pour aider les pilotes dans leur navigation, la plupart des appareils permettant le vol aux instruments sont équipés d'un système de gestion de vol appelé en anglais *Flight Management System*

(FMS). Ce système de gestion de vol est le calculateur principal des aéronefs modernes. Il centralise l'ensemble des informations sur l'aéronef, assure la navigation des appareils en fonction des moyens de repérage disponibles, permet d'optimiser vitesses et niveaux de croisière en fonction de paramètres météorologiques et économiques, et peut envoyer au pilote automatique et aux régulations moteurs toutes les informations nécessaires à l'exécution du plan de vol (nous expliquerons ce terme un peu plus loin).

Afin d'augmenter la sûreté, d'autres systèmes de positionnement que les balises de navigation sont utilisés dans de nombreux avions, comme le système de géolocalisation par satellite GPS (de l'anglais *Global Positioning System*). La possibilité de pouvoir définir des points de report quelconques a permis de rendre les routes aériennes mieux adaptées aux besoins du trafic aérien. Cependant il n'est pas possible de définir une route à travers n'importe quelle partie de l'espace aérien. Il faut éviter les zones d'espaces réservées, notamment aux militaires et respecter les contraintes environnementales.

Pour connaître la route suivie par un aéronef, il est essentiel d'avoir son altitude. À partir d'une altitude de transition qui se situe souvent à 5000 pieds au dessus du niveau du terrain, on ne parle plus d'altitude, mais de niveau de vol. Le niveau de vol, en anglais *fleight level* et d'acronyme FL, est exprimé en centaines de pieds. C'est une surface isobare liée à une pression de référence (1013,2 hPa) et séparée des autres niveaux de vol par un intervalle de pression. Par exemple, un aéronef volant au niveau de vol FL300 est à une altitude de 30 000 pieds, soit environ 9 150 mètres.

Lors d'un vol aux instruments, un pilote doit suivre une route aérienne passant par des balises standards. Dans ce cas, le pilote doit déposer un *plan de vol* avant de partir, celui-ci comporte toutes les indications du chemin qu'il va suivre, depuis son aérodrome de départ jusqu'à son aérodrome d'arrivée. Le plan de vol contient entre autres les informations suivantes :

- les aérodromes d'arrivée et de départ ;
- l'heure de départ ;
- la route aérienne prévue.

Si les pilotes volant aux instruments sont obligés de déposer un plan de vol, il est recommandé aux pilotes de vol à vue de le faire lorsqu'ils s'éloignent de leur aérodrome de départ, bien que ce ne soit pas obligatoire pour eux, sauf dans de rares cas.

Les informations fournies par le plan de vol sont utilisées par les autorités de régulation aérienne pour assurer une bonne gestion des flux de trafic et informer les contrôleurs aériens de l'arrivée des aéronefs. Les pilotes peuvent librement choisir la route qu'ils veulent suivre, bien que dans le cas de l'aviation civile commerciale la décision vienne en général des compagnies elles-mêmes. La quasi totalité des vols commerciaux déposent un plan de vol.

5.1.2 Le rôle du contrôleur aérien et la sectorisation de l'espace aérien

Selon la zone d'espace dans laquelle vole un aéronef, les services qui lui sont rendus ne sont pas identiques. L'OACI[1] a créé 7 catégories (nommées de A à G) de volumes d'espaces aériens, selon les services rendus par l'organisme de contrôle (service de contrôle, service d'information de vol et service d'alerte) et le type de vols pouvant traverser ces espaces [DGA07]. Par exemple, un volume de classe A est réservé aux vols aux instruments et le service de contrôle rendu concerne la séparation des aéronefs. D'un autre côté, dans un volume de classe E, des vols aux instruments contrôlés et des vols à vue non contrôlés se côtoient.

Malgré la diversité des zones d'espace, la grande majorité de l'espace aérien européen qui se situe au dessus de 5000 pieds est contrôlée. De plus, toutes les zones entourant les grands aéroports le sont

[1] Organisation de l'Aviation Civile Internationale, établie par la convention de Chicago de 1944 relative à l'aviation civile internationale

aussi. Assurer le contrôle de tous ces espaces, nécessite un grand nombre de contrôleurs aériens, sans compter que beaucoup d'espaces non contrôlés disposent de services d'informations de vol nécessitant aussi des contrôleurs.

La définition officielle du contrôle du trafic aérien est la suivante :
Le but premier du contrôle du trafic aérien est d'assurer la sécurité du trafic et donc d'éviter les abordages[2] *entre aéronefs opérant dans le système, puis d'optimiser les flux de trafic.*

Le contrôle aérien se divise en trois catégories :

- **le contrôle au sol** correspond au contrôle des aéronefs sur une partie des plates-formes aéroportuaires, ce sujet est bien présenté dans [GOT04] ;
- **le contrôle d'approche** s'occupe des zones entourant les plates-formes aéroportuaires. Le but de ce contrôle est d'organiser la descente des aéronefs en vue de leur atterrissage, ainsi que leur montée, après leur décollage ;
- **le contrôle en route** représente la face la moins visible du contrôle aérien. Cependant, c'est aussi la phase de contrôle de loin la plus longue pour un vol commercial, et qui nécessite le plus grand nombre de contrôleurs. Le contrôle en route consiste à amener les aéronefs depuis la sortie de la zone de contrôle d'approche vers leurs altitudes de croisière, puis à les faire évoluer le plus conformément possible par rapport à leurs plans de vol vers leurs aérodromes de destination où ils seront de nouveau confiés, après une phase de descente, au contrôle d'approche.

Afin d'éviter les abordages, chacun de ces contrôles agit sur la séparation des aéronefs. Il existe des normes de séparation horizontales et verticales dépendant, entre autres, de l'endroit où sont contrôlés les aéronefs, ainsi que des types d'aéronefs à séparer. Dans chacune de ces catégories de contrôle, le contrôleur aérien est chargé de faire respecter ces normes entre les aéronefs, et pour cela il peut leur donner de nouvelles trajectoires.

Le contrôle en route ne se fait pas dans les tours de contrôle des aéroports. Il est effectué par des contrôleurs regroupés en centres de contrôle en route. À titre d'exemple, il existe 5 centres de contrôle en route en France répartis dans les villes suivantes : Aix-en-Provence, Bordeaux, Brest, Paris et Reims. Le tableau 5.1 présente la liste des centres de contrôle de plusieurs pays européens.

Les contrôleurs aériens du contrôle en route surveillent un espace limité, appelé secteur de contrôle en route. Par la suite, nous utiliserons l'abréviation *secteur* pour désigner un secteur de contrôle en route. Ces secteurs sont caractérisés par leurs niveaux de vol et leurs frontières, ces dernières pouvant changer à chaque niveau de vol. La figure 5.2 représente la projection d'un des secteurs de l'espace aérien français, au dessus de Marseille. Ce secteur nommé LFMM1 est localisé entre les niveaux de vol FL195 et FL305. Il existe un secteur, LFMM2, situé au dessus de lui, et un secteur, LFMML, en dessous.

La taille des secteurs est adaptée au trafic que le contrôleur sera amené à surveiller. En effet, la charge de contrôle que peut subir un humain est limitée. Le seuil au delà duquel la charge de contrôle est trop importante relève souvent du cas par cas, en fonction des routes empruntées par l'avion. Cependant, un seuil « classique » limite à 30 le nombre d'avions par heure dans un secteur, avec un pic de 15 avions pendant au maximum 10 minutes. Cette contrainte explique entre autres les différences de tailles entre les secteurs et leurs formes.

Le trafic aérien étant très variable au cours de la journée, pour s'adapter à celui-ci, la taille des secteurs change. Il existe donc des regroupements de secteurs aux périodes creuses (pendant la nuit), et des périodes où les secteurs sont de petites tailles pour répondre à la demande de trafic. Ainsi, on fait la distinction entre secteur élémentaire et secteur regroupé, ce dernier constituant le regroupement de plusieurs secteurs élémentaires. Le tableau 5.1 recense le nombre de secteurs élémentaires présents

[2] Il y a abordage entre deux aéronefs lorsque ceux-ci rentrent en contact quand ils volent.

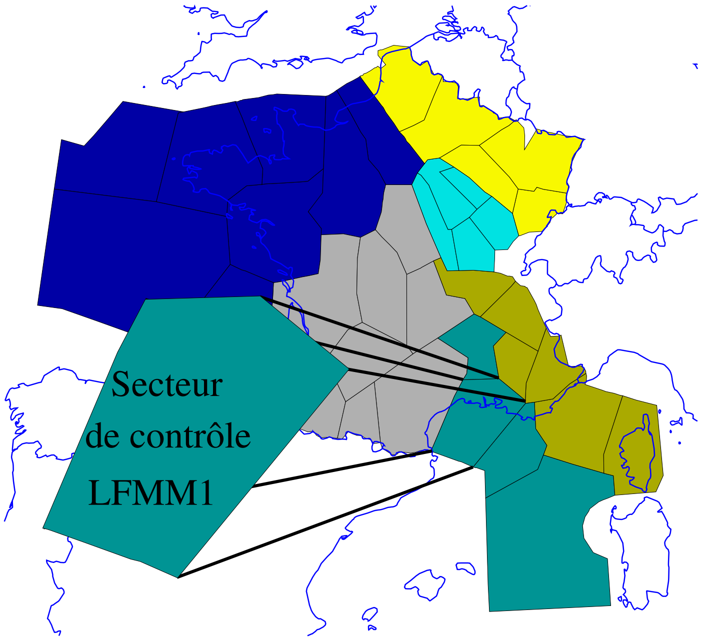


FIG. 5.2 – Projection du secteur de contrôle LFMM1, au niveau de vol FL300, en France, au dessus de Marseille.

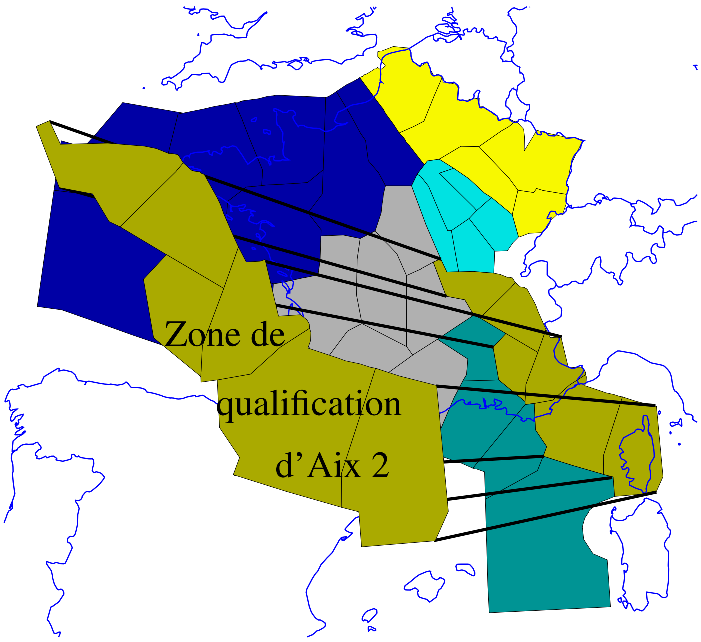


FIG. 5.3 – Projection de la seconde zone de qualification d'Aix en Provence au niveau de vol FL300.

dans différents centres de contrôle en Europe. Dans cette étude, nous ne nous intéresserons qu'aux secteurs élémentaires, qui sont les « briques élémentaires » à la base du contrôle aérien.

Pour effectuer le contrôle d'un secteur, le contrôleur doit obtenir une qualification. Un contrôleur n'est pas formé pour le contrôle d'un unique secteur, mais pour pouvoir prendre en charge différents secteurs de manière alternative. L'ensemble des secteurs sur lesquels il est habilité à travailler est appelé zone de qualification. La figure 5.3 représente la seconde zone de qualification du centre de contrôle en route d'Aix en Provence. Il est facile de comprendre que les zones de qualification sont intimement liées aux centres de contrôle en route. En effet, il n'est pas en pratique possible de former un contrôleur sur des secteurs contrôlés à des endroits totalement différents. Ainsi, il peut y avoir une ou plusieurs zones de qualification par centre de contrôle. À titre d'exemple, la France possède 7 zones de qualification, 2 à Aix-en-Provence et Paris et une à Bordeaux, Brest, Paris et Reims. Le tableau 5.1 présente la liste des zones de qualification des pays de la *core area* européenne (définie ci-dessous), plus l'Irlande.

La *core area* est définie comme la région du ciel aérien européen de plus forte densité de trafic. Cette notion est souvent utilisée par Eurocontrol dans ses exemples, mais sans jamais définir les pays ou centres de contrôle qu'elle contient. Confronté à ce problème pendant notre stage de master recherche, nous avons voulu définir avec une approche scientifique quelle région du ciel européen pouvait être la *core area*. L'étude que nous avons réalisée est présentée dans [BIC04a, BIC05]. Après avoir utilisé une méthode s'inspirant de la mécanique des fluides et utilisant les nombres de Reynolds, nous avons trouvé un ensemble de pays qui concentrait la plus forte densité de trafic en Europe : l'Espagne, l'Italie, la Suisse, l'Autriche, l'Allemagne, la Belgique, le Luxembourg, les Pays-Bas, le Danemark, la France et le Royaume-Uni.

Pour des raisons pratiques, le contrôle en route différencie l'espace supérieur de l'espace inférieur. Dans la réglementation du Parlement européen, « le niveau de division entre les espaces aériens supérieur et inférieur est fixé au niveau de vol 285 » (8 687m) [RÈG04c]. Cependant, de manière opérationnelle, on situe l'espace supérieur au dessus du niveau de vol FL195 (5 944m) et l'espace inférieur en dessous. Cette différentiation opérationnelle est basée sur le niveau de vol de croisière des avions commerciaux. En effet, celui-ci se situe au dessus du niveau de vol FL195. Or quand les avions ne sont pas en vol de croisière, ils sont en phases de montée ou de descente. Le contrôle qui en résulte et les problèmes qui y sont liés sont donc différents. Malgré l'intérêt de la différenciation opérationnelle, nous utiliserons par la suite la différenciation introduite par la réglementation européenne.

Le lecteur pourra aller voir en annexe B une présentation plus approfondie du travail des contrôleurs aériens, ainsi qu'une présentation sur la régulation actuelle du trafic aérien. Cependant, cette lecture n'est pas utile à la compréhension du reste de cette thèse.

Pays (Code OACI) Centre	Nb. de centres Code centre	Nb. de zones	Nb. de secteurs
Eurocontrol[3]	1 centre	3 zones	20 secteurs
Maastricht	EDYY	3	20
Royaume-Uni (EG)	3 centres	9 zones	71 secteurs
Manchester	EGCC	2	8
Écosse	EGPX	1	18
Londre	EGTT	6	45
			Fin du tableau à la page suivante.

[3] Eurocontrol est l'organisation européenne pour la sécurité de la navigation aérienne.

Pays (Code OACI) Centre	Nb. de centres Code centre	Nb. de zones	Nb. de secteurs
Belgique (EB)	1 centre	1 zone	7 secteurs
Bruxelles	EBB	1	7
Allemagne (ED)	6 centres	11 zones	103 secteurs
Berlin	EDBB	1	15
Frankfort	EDFF	4	19
Düsseldorf	EDLL	3	10
Munich	EDMM	1	20
Karlsruhe	EDUU	1	27
Bremen	EDWW	1	12
Pays-Bas (EH)	1 centre	2 zones	10 secteurs
Amsterdam	EHAA	2	10
Irlande (EI)	2 centres	2 zones	44 secteurs
Dublin	EIDW	1	2
Shannon	EISN	1	42
Danemark (EK)	1 centre	2 zones	14 secteurs
Copenhague	EKDK	2	14
Luxembourg (EL)	0 centres	0 zones	0 secteurs
Espagne (LE)	4 centres	5 zones	75 secteurs
Barcelone	LECB	1	21
Madrid	LECM	2	26
Palma	LECP	1	7
Séville	LECS	1	21
France (LF)	5 centres	7 zones	139 secteurs
Bordeaux	LFBB	1	31
Reims	LFEE	1	13
Paris	LFFA	2	38
Aix	LFMA	2	36
Brest	LFRR	1	21
Italie (LI)	4 centres	5 zones	117 secteurs
Brindisi	LIBB	1	13
Milan	LIMM	1	17
Padoue	LIPP	1	37
Rome	LIRR	2	50
Autriche (LO)	1 centre	1 zone	17 secteurs
Vienne	LOVV	1	17
Suisse (LS)	2 centres	2 zones	21 secteurs
Genève	LSAG	1	11
Zurich	LSAZ	1	10
Total :	31 centres	50 zones	638 secteurs

TAB. 5.1: Répartitions des secteurs, zones et centres de contrôle en juin 2005 dans les 11 pays de la *core area* européenne, et l'Irlande.

5.2 Le problème du découpage de l'espace aérien

Comme nous l'avons vu dans la section précédente (*supra* 5.1), l'espace aérien est divisé en zones de qualification faisant elles-mêmes partie de centres de contrôle en route. Il existe donc déjà actuellement plusieurs découpages de l'espace aérien en Europe, selon que l'on parle du découpage en secteurs, en zones ou en centres. Le tableau 5.1 de la section précédente donne quelques caractéristiques des ces différents découpages.

L'étude du découpage de l'espace aérien européen qui nous a été demandée ne concerne pas le redécoupage des secteurs de contrôle. Plusieurs travaux ont déjà été réalisés sur ce sujet [DEL95, DAC04]. Cependant, toute la partie théorique sur le partitionnement de graphe des chapitres 2 à 4 pourrait certainement être avantageusement adaptée à ce problème.

L'étude qui nous a été demandée concerne la réorganisation des ensembles de secteurs de contrôle en zones de qualification, dans la vision de ce que sera le ciel aérien européen de demain, c'est-à-dire avec la création de blocs d'espaces fonctionnels voulue par la Commission européenne. Nous verrons dans la sous-section suivante en quoi consiste la création de ces blocs d'espaces fonctionnels.

Les secteurs de contrôle concernés par notre étude sont ceux des zones de qualification, il s'agit donc des secteurs des espaces supérieurs et inférieurs des centres de contrôle en route. À partir des zones de qualification qui seront proposées, il sera dans une autre étude possible de créer les centres de contrôle en route correspondant, ceux-ci étant des regroupements d'une ou plusieurs zones de qualification.

5.2.1 La création de blocs d'espace aérien fonctionnels en Europe

La création de blocs d'espace aérien fonctionnels (en anglais, *functional airspace block* ou FAB) en Europe s'inscrit dans le cadre du projet de *Ciel unique européen*. La Commission européenne a adopté le 10 octobre 2001 un ensemble de mesures sur la gestion du trafic aérien dans le but de pouvoir élaborer le cadre réglementaire du ciel unique européen fin 2004. L'objectif du ciel unique européen est de mettre fin à la fragmentation de l'espace aérien de l'Union européenne et de créer un espace sans frontières, sûr et efficace. Il a résulté de ces travaux un règlement cadre [RÈG04a] et trois règlements associés [RÈG04b, RÈG04c, RÈG04d]. Ces règlements prévoient la création de *blocs d'espace aérien fonctionnels* (*Fonctional airspace block* en anglais) dans l'espace aérien supérieur, au dessus de l'Union européenne. Dans le règlement cadre, un bloc d'espace aérien fonctionnel (qui sera noté par la suite bloc fonctionnel) est défini comme : « un bloc d'espace aérien fondé sur des besoins opérationnels, traduisant la nécessité d'assurer une gestion plus intégrée de l'espace aérien, indépendamment des frontières existantes. »

Les règlements insistent sur le fait que la reconfiguration de l'espace aérien devra être fondée sur des besoins opérationnels, indépendamment des frontières existantes. D'autre part, les principes généraux liés à la création des blocs fonctionnels devront être élaborés par les prestataires de service de la navigation aérienne des différents États membres en concertation avec Eurocontrol et sur la base de ses conseils techniques. La création et la modification des blocs fonctionnels sont quant à eux du ressort des États membres concernés. Le règlement prévoit que les blocs fonctionnels devront être mis en place d'ici 2009.

Concernant la reconfiguration de l'espace aérien supérieur (au dessus du FL285) en blocs fonctionnels, deux conditions nécessaires à leurs créations touchent plus particulièrement notre application au découpage de l'espace aérien :

– les blocs fonctionnels « permettent une utilisation optimale de l'espace aérien, compte tenu des courants de trafic aérien ». La gestion des courants de trafic aérien a pour « but de contribuer à

un écoulement en toute sécurité, ordonné et rapide du trafic aérien » ;
– les blocs fonctionnels « sont justifiés par la valeur ajoutée globale qu'ils apportent ».

Afin d'établir les principes généraux liés à la création de blocs fonctionnels, des séminaires ont été organisés par Eurocontrol en invitant les différents prestataires de service de la navigation aérienne de l'Union européenne, dont la DSNA[4] fait partie. Ainsi, trois réunions de consultation ont eu lieu entre 2004 et début 2005. Nous avons pu participer à la dernière d'entre elles au tout début de cette thèse. Chacune de ces réunions avait pour but la validation du travail réalisé par différentes équipes sur les problèmes liés à la création et au fonctionnement de blocs fonctionnels. Un rapport consultatif pour la Commission européenne résume le travail accompli et concrétisé par ces séminaires [EUR05].

Il a été retenu pendant ces séminaires que les blocs d'espace fonctionnels devaient [EUR05] :
– être dessinés sur la base de besoins opérationnels ;
– créer un espace de navigation plus cohérent ;
– ne pas tenir compte des frontières nationales.

Les blocs d'espace fonctionnels doivent être dessinés pour optimiser les flots de trafic aérien. Mais il a aussi été admis que la création de blocs d'espace fonctionnels était un processus complexe et très long.

Dans [RÈG04c], il est précisé que les blocs fonctionnels doivent respecter sept critères :

1. les FAB sont étayés par un dossier de sécurité ;
2. ils permettent une utilisation optimale de l'espace aérien, compte tenu des courants de trafic aérien ;
3. ils sont justifiés par la valeur ajoutée globale qu'ils apportent, y compris l'utilisation optimale des ressources techniques et humaines, sur la base d'analyses coût/efficacité ;
4. ils assurent un transfert fluide et souple de la responsabilité du contrôle de la circulation aérienne entre les unités des services de la circulation aérienne ;
5. ils assurent la compatibilité entre les configurations des espaces aériens supérieur et inférieur ;
6. ils respectent les conditions découlant des accords régionaux conclus au sein de l'OACI, et
7. ils respectent les accords régionaux qui existent à la date d'entrée en vigueur du présent règlement [...]

Ces critères sont soumis à interprétation. Nous présentons dans ce document la notre, mais d'autres interprétations peuvent être intéressantes. Nous avons essayé de modéliser ces critères le mieux possible afin de les prendre en compte. Cependant, ces critères sont soit trop généraux pour être parfaitement modélisés (comme les critères 2, 3, 4 et 5), soit tout simplement pas modélisables dans le cadre de notre problème (c'est le cas des critères 1, 6 et 7).

Des structures similaires aux blocs fonctionnels existent déjà en Europe. C'est par exemple le cas de la zone d'espace aérien contrôlée par le centre de contrôle d'Eurocontrol à Maastricht. Ce centre est chargé du contrôle de l'espace aérien supérieur du Bénélux et du nord-ouest de l'Allemagne depuis 1972. C'est aussi le cas depuis 1997 dans l'est de l'Europe avec la création du *Central European Air Traffic Services* qui regroupe les espaces aériens de l'Autriche, la Hongrie, la République Tchèque, la Slovaquie, la Slovénie, la Croatie, la Serbie et le nord-est de l'Italie.

La structure officielle et multi-nationale des blocs fonctionnels va pouvoir faciliter et généraliser la démarche de création de zones de contrôle communes comme à Maastricht ou dans l'est de l'Europe, et assurer leur pérennisation.

[4] Direction de services de la navigation aérienne, organisme commanditaire de cette thèse.

Cette étude a pris en compte tous les secteurs des centres de contrôle en routes, qu'ils soient dans les espaces supérieur ou inférieur. Bien que la réglementation ne prévoie pour l'instant la création des blocs fonctionnels que dans l'espace supérieur, nous avons pris ce parti pour trois raisons :

– peu de zones de qualification actuelles sont découpées en fonction du niveau de vol FL285, et changer de zone de qualification au FL285 peut être néfaste pour la fluidité du trafic ;
– cette étude n'ayant pas pour but de proposer une solution opérationnelle immédiate, comme « qui peut le plus, peut le moins », il nous a semblé plus judicieux de chercher à résoudre le problème de plus grande taille ;
– le règlement [RÈG04c] prévoit aussi dans son article 10 la possibilité d'étendre les blocs fonctionnels aux espaces inférieurs. Or, il semble que cette possibilité soit celle envisagée par la plupart des prestataires de services de la navigation aériennes des états européens. Dans ce cas, d'après le règlement, les blocs fonctionnels engloberaient tous les secteurs des centres de contrôle en routes, sans distinctions.

5.2.2 La création d'un bloc fonctionnel en Europe centrale

Une étude de faisabilité est en cours concernant la création d'un bloc fonctionnel au cœur de l'Europe : le *FAB d'Europe centrale*. Six États se sont associés pour ce projet : l'Allemagne, la Belgique, la France, le Luxembourg, les Pays-Bas et la Suisse. Cette étude fait suite au rapprochement entre les projets de blocs fonctionnels de la France et de la Suisse, ainsi que de l'Allemagne et du Bénélux. Ces deux projets de blocs fonctionnels avaient été préparés entre 2005 et 2006.

Le but du FAB d'Europe centrale est de mettre en place un bloc fonctionnel englobant l'espace aérien supérieur et inférieur de ces 6 États, ainsi que des solutions communes axées sur la performance et indépendantes des frontières nationales [FAB07]. La zone couverte par ce bloc fonctionnel est l'une des plus denses au monde en terme de trafic et se caractérise par des routes aériennes civiles et militaires étroitement imbriquées.

Les maîtres mots de ce projet sont « continuité opérationnelle » et « augmentation de la capacité ». L'étude de faisabilité devra se terminer en 2008 afin de respecter la date de création des blocs fonctionnels prévue pour 2009. Les avantages attendus de tels blocs fonctionnels sont les suivants :

– l'absorption intégrale de la croissance escomptée du trafic ;
– l'utilisation optimale de l'espace aérien suivant les flux de trafic ;
– l'exploitation optimale des ressources humaines et techniques ;
– une parfaite compatibilité entre les espaces aériens inférieur et supérieur ;
– un niveau de service économiquement performant ;
– une coopération civile-militaire renforcée.

La démarche entamée par la création du FAB d'Europe centrale pourrait inclure un refonte de l'espace aérien civil et militaire indépendamment des frontières nationales.

Ainsi, toute création de blocs fonctionnels est susceptible de s'accompagner d'un redécoupage de l'espace aérien sans tenir compte des frontières nationales. Comme dans le cas de la sous-section précédente, les secteurs concernés par ce redécoupage sont ceux des centres de contrôle en routes des différents pays concernés.

5.3 Modélisation du problème

Cette thèse a pour objet d'aider techniquement à la réalisation du redécoupage de l'espace aérien dans le cadre de la création de blocs fonctionnels, en proposant des outils informatiques de redécoupage. Or, comme nous l'avons vu dans la sous-section précédente, les blocs fonctionnels doivent être créés dans un but d'utilisation optimale de l'espace aérien suivant les flux de trafic et d'exploitation optimale des ressources humaines et techniques. Pour répondre à ces impératifs, il faut s'intéresser à la charge de contrôle d'un contrôleur dans un secteur.

5.3.1 Charge de contrôle dans un secteur

Un secteur est contrôlé par une paire de contrôleurs qui assurent la sécurité des vols qui y transitent en séparant les aéronefs entre eux selon les normes en vigueur. Plus le nombre d'avions dans un secteur est important, plus la charge de contrôle induite augmente, et ce de façon non linéaire. Il existe une limite au delà de laquelle le contrôleur ne peut plus accepter de nouveaux avions et oblige ceux-ci à passer par des secteurs voisins moins chargés. On dit alors que le secteur est saturé. Cet état critique doit être évité car il provoque un phénomène cumulatif de surcharge sur les secteurs amonts pouvant remonter jusqu'aux aéroports de départ. Après une enquête auprès des contrôleurs [DEL95, ALL96], on remarque que la charge de travail dans un secteur dépend de critères quantitatifs et qualitatifs. Ces derniers regroupent essentiellement les facteurs humains dont le principal est le stress. Bien évidemment, comme tous les contrôleurs ne réagissent pas de la même manière à une situation de trafic complexe, il est difficile de fournir un modèle mathématique de stress applicable aux contrôleurs. L'enquête a cependant permis de relever les critères quantitatifs suivants :

- **la charge de conflit :** deux avions sont dit en conflit lorsque la distance qui les sépare risque de devenir inférieure à la norme de séparation en vigueur entre ces deux catégories d'avions. Lorsque deux avions sont en conflit, le contrôleur peut modifier la route de ceux-ci, ou encore leurs niveaux de vol, afin d'assurer le respect de la norme de séparation ;
- **la charge de coordination :** dans un même secteur, tous les avions communiquent avec le contrôleur sur la même fréquence. Lorsqu'un avion change de secteur, le contrôleur qui l'avait en charge opère un transfert de contrôle et l'avion doit changer de fréquence. Au préalable, le transfert doit avoir été accepté par le contrôleur qui reçoit l'avion. Une négociation entre les deux contrôleurs, celui qui reçoit et celui qui transmet, a lieu pour s'assurer que l'avion peut être accepté et définir les modalités de transfert comme le niveau de vol, le cap, *etc*. Chaque transfert nécessite un travail relativement important de la part des contrôleurs. De plus, c'est une opération au cours de laquelle des incompréhensions ou des erreurs peuvent se produire causant des pertes accidentelles de séparation. Dans un réseau de transport sectorisé, la charge de coordination est proportionnelle aux flux coupés par les frontières des secteurs ;
- **la charge de monitoring :** dans un secteur, les avions qui ne sont pas en conflit ou en transfert nécessitent une surveillance de la part du contrôleur. En effet, celui-ci vérifie le bon déroulement des plans de vol sur l'image radar et essaie de déterminer les risques potentiels de conflits futurs induits par ces avions. Le monitoring est la tâche de fond du contrôleur mais qui représente une source importante de stress pour celui-ci. Cette charge est directement liée au nombre d'avions dans le secteur.

Il existe d'autres charges de contrôle facilement quantifiables [TUA76], mais leur impact sur l'ensemble de la charge de contrôle d'un secteur est négligeable par rapport aux trois précédentes.

5.3.2 Un objectif : minimiser la charge de travail des contrôleurs

Il est connu que la coordination est bien plus simple et efficace entre secteurs d'une même zone de qualification qu'entre zones différentes [HAL05]. Il y a plusieurs causes à cela. La première est qu'un contrôleur qui transmet un avion vers un secteur de sa zone de qualification, connaît parfaitement ce dernier et notamment les procédures et habitudes le concernant. En effet, il est régulièrement amené à contrôler ce secteur. D'autre part, le contrôleur qui recevra l'avion se situe souvent dans la même pièce que le premier, ce qui peut notablement faciliter les échanges en cas de problème. Les autres facteurs sont nombreux, comme l'absence de problème linguistique (ce qui peut arriver entre deux pays limitrophes), un système de communication beaucoup plus fiable (la distance, comme les différences d'opérateurs de communication, rendent celle-ci moins sûre), la connaissance des deux contrôleurs et l'habitude de travailler ensemble, *etc*. Ainsi, la charge de coordination subie par un contrôleur est moins importante dans le cas des deux secteurs d'une même zone de qualification que dans le cas de deux secteurs de zones différentes.

De ce que nous venons d'expliciter, il apparaît deux objectifs permettant de diminuer la charge de coordination des contrôleurs :

1. minimiser la charge de coordination entre secteurs de zones de qualification différentes. Pour cela, la solution la plus efficace est sans conteste de minimiser le nombre d'aéronefs passant entre les différentes zones de qualification ;
2. maximiser le nombre d'aéronefs entre secteurs d'une même zone. La coordination étant beaucoup plus simple et efficace entre secteurs d'une même zone, cela devrait permettre aux contrôleurs de prendre en charge plus d'avions. En effet, pour un nombre de coordinations constant, plus de coordinations entre secteurs d'une même zone diminue la charge de coordinaiton, ce qui permet potentiellement de prendre en charge plus d'avions.

Ainsi, s'il semble possible de minimiser la charge de coordination des contrôleurs par un redécoupage des zones de qualification, il paraît impossible de minimiser les charges de conflit et de monitoring. Cependant, la diminution du nombre d'aéronefs passant entres les zones de qualification et l'augmentation de ce nombre entre secteurs d'une même zone fait que les coordinations sont mieux réalisées. En effet, si l'on tient compte du fait que le contrôleur du secteur de départ connaît bien les procédures et habitudes du secteur d'arrivée et qu'il a une visualisation sur son écran radar d'une partie du secteur d'arrivée, l'avion sera acheminé vers le second secteur de façon à minimiser la charge de conflit du contrôleur du secteur d'arrivée. Ainsi, la minimisation de la charge de coordination se répercute sur la minimisation de la charge de conflit.

Quant à la charge de monitoring, il ne semble pas qu'elle soit réellement affectée par une diminution de la charge de coordination. En effet, la diminution de cette dernière n'a pas pour objet de diminuer le nombre d'avions surveillés dans le secteur, seul critère qui affecte la charge de monitoring.

5.3.3 Deux contraintes : la taille des zones de qualification et celle des centres de contrôle

Plusieurs discussions avec quelques contrôleurs aériens français ont permis de situer le nombre optimal de secteurs par zones de qualification entre 22 et 28 secteurs. Les chiffres ainsi avancés sont subjectifs et mériteraient une étude approfondie pour être validés. Cependant, ces chiffres nous semblent importants car ils viennent de professionnels du secteur.

Le nombre moyen de secteurs par centre de contrôle est actuellement de 9 pour les 38 pays membres d'Eurocontrol[5]. Ce nombre est de 37 aux États-Unis. Des études ont été réalisées qui

[5] Les 38 pays membres d'Eurocontrol sont en mai 2007 : l'Albanie, l'Allemagne, l'Arménie, l'Autriche, la Belgique, la Bosnie Herzégovie, la Bulgarie, la Croatie, Chypre, la République Chèque, le Danemark, l'Espagne, la Finlande, la France, la Grèce, la Hongrie, l'Irlande, l'Italie, la Lithuanie, le Luxembourg, la Macédoine, Malte, la Moldavie, Monaco, le Monténégro, la Norvège, les Pays-Bas, la Pologne, le Portugal, la Roumanie, le Royaume-Uni, la Serbie, la Slovaquie, la Slovénie, la Suède, la Suisse, la Turquie et l'Ukraine.

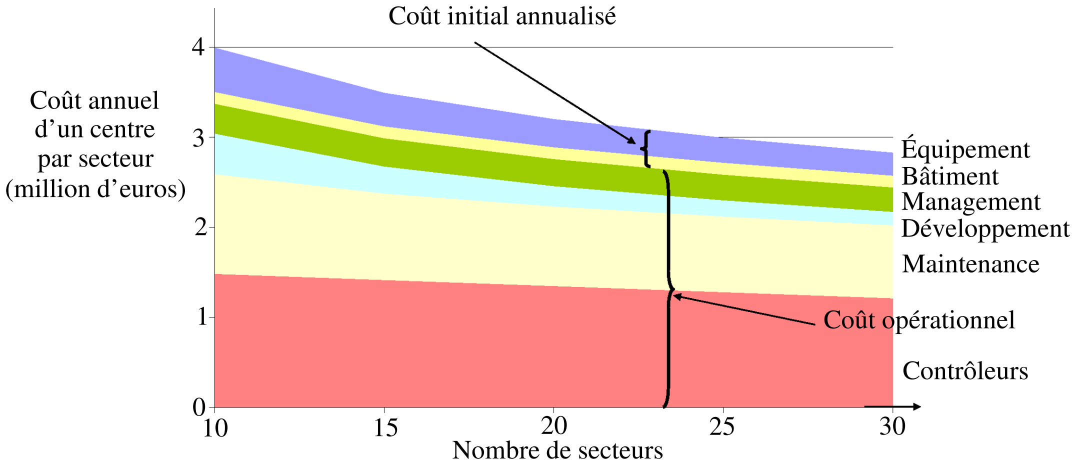


FIG. 5.4 – Variation du coût par secteur en fonction de la taille du centre [EUR06a].

pointent du doigt ces divergences [EUR06a, EUR06b]. Dans un but d'optimisation des coûts, elles préconisent la création de centres de contrôle avec un nombre optimal de 25 secteurs par centre. Selon ces mêmes études, ce nombre ne devrait pas dépasser 50 à 60 secteurs pour un même centre, seuil au-delà duquel les économies d'échelle sont rattrapées par des coûts liés à la taille du centre. La figure 5.4 représente la variation du coût par secteur en fonction de la taille du centre. En adoptant une configuration de 25 secteurs par centre, les économies prévues par rapport au système actuel sont de l'ordre de 20% pour les coûts d'exploitation et de 25% pour les coûts d'investissement.

Dans la mesure où la création des blocs fonctionnels prévue par la commission doit être basée sur une analyse coût/efficacité [RÈG04c] (*supra* 5.2), nous considérons que la taille de 25 secteurs sans dépasser 50 secteurs, est une contrainte forte de notre problème. Nous ferons passer en priorité les recommandations d'Eurocontrol devant celles que nous avons reçues des quelques contrôleurs que nous connaissons, car ces dernières sont beaucoup plus discutables sur leurs fondements scientifiques que l'analyse faite par Eurocontrol.

Cependant, il est remarquable de constater les similitudes sur le nombre de secteurs par zone de qualification proposés par ces contrôleurs aériens et ceux par centres résultant d'analyses économiques. Nous retiendrons donc qu'une zone de qualification doit de préférence comporter 25 secteurs et ainsi correspondre à un centre mais que, cependant, en accord avec les recommandations d'Eurocontrol, ce chiffre offre une forte tolérance.

Ainsi, il apparaît qu'une bonne configuration de l'espace aérien consiste à avoir des centres de contrôle possédant de préférence une zone de qualification d'environ 25 secteurs. Et il faut que la taille d'un centre ne dépasse pas 50 secteurs. Comme un centre de contrôl contient en au moins une zone de qualification, chaque zone de qualification ne devra pas posséder plus de 50 secteurs.

Un autre critère important est la charge de contrôle de chaque zone. Une zone de qualification peut accueillir plus de secteurs si le nombre moyen d'avions par secteur n'est pas élevé. D'un autre côté, si ce nombre moyen d'avions est élevé, la zone de qualification devra être de taille réduite. En effet, lorsque le nombre d'avions augmente, le nombre de routes et/ou de procédures à connaître augmente aussi. Dans ces cas-là, les contrôleurs doivent contrôler plus souvent chaque secteur pour rester efficace sur ceux-ci, ce qui veut dire, à charge de travail égal, qu'ils en connaîtront moins. Ainsi, il est très important de garder un certain équilibre entre les différentes zones au niveau du nombre d'avions traversant chacune d'entre elles. De plus, un déséquilibre trop important de charge de contrôle entre les zones constituerait un préjudice pour les contrôleurs de la zone la plus chargée.

Maintenant que l'on connaît les objectifs et les contraintes liés à la minimisation de la charge de travail des contrôleurs, il nous faut analyser et traiter les données dont nous disposons afin de pouvoir les utiliser.

5.3.4 Analyse et traitement des données du trafic aérien européen

Comme nous l'avons vu au début de ce chapitre (*supra* 5.1), les pilotes IFR (vol aux instruments) doivent déposer un plan de vol avant leur départ. En Europe, les plans de vol ont un traitement centralisé qui dépend d'Eurocontrol (*infra* annexe B). Nous avons pu récupérer auprès d'Eurocontrol 3 journées de trafic de juin 2005 (les vendredi 17 et 24 juin, ainsi que le lundi 20 juin) et une centaine de jours répartis entre juillet 2001 et septembre 2002. Ces dernières données sont un peu anciennes et moins représentatives du trafic actuel que celles de 2005 qui font suite aux attentats du 11 septembre 2001. Nous utiliserons donc autant que possible les données de 2005, et particulièrement celle du vendredi 17 juin, même si cette « pénurie » de données ne nous permet pas des conclusions aussi précises qu'avec les données d'une année entière de trafic. Cependant, comme nous l'avons déjà précisé, notre but est simplement de fournir des outils permettant une redécoupage à partir de données actuelles.

Les données de plan de vol sont fournies par journée UTC[6] dans des fichiers récapitulant les numéros de vol et les horaires de passage des avions dans les secteurs traversés avec les noms de ces derniers. Les données plan de vol sont fournies avec un descriptif des coordonnées en 3 dimensions de chaque secteur. Ces coordonnées, parfois inexactes ou incomplètes, nous ont permis, après traitement, de créer des cartes des secteurs et des zones de qualification aux dates correspondantes.

Le but de l'analyse des plans de vol européens est d'obtenir une base de données quotidienne des flux d'avions entre les différents secteurs européens. Ainsi, pour chaque secteur en Europe, on connaît le nombre d'aéronefs ayant déposé un plan de vol qui l'ont traversé, ainsi que les flux entre ce secteur et chacun de ses voisins.

Les données de plan de vol fournies par Eurocontrol ont dû être longuement traitées avant d'être utilisées. Ce document n'a pas pour objet de détailler les étapes de ce traitement de données, cependant, nous devons souligner quelques grosses incohérences auxquelles notre programme de traitement de données a dû faire face. Le problème le plus fréquemment rencontré concerne la désignation des secteurs. En effet, de nombreux secteurs présents dans les plans de vol n'apparaissent pas dans la nomenclature des secteurs associée. Un autre problème récurrent concerne la validité des horaires inscrits. Ainsi, certains vols « remontent le temps » par endroits.

Un premier outil de traitement des plans de vol a été réalisé pendant mon stage de master recherche [BIC04a]. Cependant cet outil a été complètement refait au début de cette thèse, en partie pour accueillir les données de 2005 qui avaient changé de format, en partie pour améliorer l'efficacité et la rapidité de ce programme. Cependant, les concepts généraux liés au traitement de données décrits dans [BIC04a] restent toujours valables.

Il nous faut préciser que la base de données des flux d'avions entre les différents secteurs ne contient pas uniquement des flux entre secteurs. En effet, certains flux n'ont pas pour origine ou destination un secteur connu. C'est le cas des aéronefs venant de décoller d'un aérodrome, ou encore celui d'un aéronef atterrissant. Mais c'est aussi le cas des aéronefs sortant de la zone couverte par le descriptif des secteurs de la base de données d'Eurocontrol. Ainsi, un aéronef se rendant au Canada

[6]UTC est l'acronyme de *United Time Coordinated* ou en français temps universel coordonné. C'est une échelle de temps internationale.

sortira d'Europe, mais sera quand même comptabilisé dans un flux sortant du dernier secteur européen qu'il aura traversé.

5.3.5 Graphe du trafic aérien européen et adaptation au partitionnement

La base de données quotidienne des flux d'avions entre les différents secteurs européens obtenue grâce à l'analyse des plans de vol européens, permet de créer un graphe du trafic aérien européen. Pour cela, il faut considérer chaque secteur de contrôle comme étant un sommet de ce graphe. De plus, à chaque flux entre deux secteurs est associée une arête du graphe entre les sommets correspondants, qui a pour poids le nombre d'avions de ce flux. Comme nous avons vu à la fin de la sous-section précédente, certains flux ne correspondent pas à des flux entre secteurs. Nous avons choisi de prendre en compte ces flux dans le poids d'un sommet. Ainsi, le poids d'un sommet correspond à la somme du nombre d'aéronefs de tous les flux liés au secteur correspondant à ce sommet. Ce qui veut donc dire que le poids d'un sommet n'est pas égal à la somme des poids des arêtes liées à ce sommet. Pour autant, le poids de ce sommet a une signification bien concrète, car il correspond au nombre de tâches de coordinations effectuées dans le secteur correspondant pendant une journée. De même, le poids d'une arête correspond au nombre de tâches de coordinations effectuées par les contrôleurs du secteur de départ ou par ceux du secteur d'arrivée. Ainsi, la base de données des plans de vol permet la création d'un graphe. Le problème du redécoupage de l'espace aérien européen devient donc un problème de partitionnement de graphe.

Nous avons vu que la minimisation de la charge de travail des contrôleurs consistait à minimiser la charge de coordination entre les zones de qualification et à maximiser cette charge entre secteurs d'une même zone. Dans l'adaptation au partitionnement de graphe, la fonction de coût (*supra* 1.4) qui semble le plus correspondre à cet objectif est la coupe normalisée. Cependant, dans la mesure où le poids d'un sommet n'est pas égal à la somme des poids des arêtes liées à ce sommet, mais correspond au nombre de tâches de coordinations effectuées dans le secteur, il est préférable d'utiliser comme fonction objectif le ratio de coupe plutôt que la coupe normalisée. C'est donc le ratio de coupe qui sera utilisé comme fonction objectif du découpage de l'espace aérien dans cette étude.

Nous considérerons que le poids d'une zone de qualification est égual à la somme des poids des secteurs, *i.e.* des sommets, la constituant.

Les contraintes de notre problème sont liées à la taille et au poids des zones de qualification. Il s'agit de contraintes sur l'équilibre de la partition. Comme nous avons vu, une zone de qualification doit posséder en moyenne 25 secteurs et au maximum 50 secteurs. Nous n'avons pas trouvé de textes concernant la variation de poids des zones de qualification. Cependant, dans la mesure où le poids d'une zone de qualification est un des critère permettant d'évaluer sa difficulté de contrôle, il nous a semblé important de limiter la balance de partitionnement sur le poids des zones de qualification à $2,0$. Cette « grande » valeure a été volontairement choisie, car le poids d'une zone est loin d'être le seul indicateur de la difficulté de contrôl, cependant à partir du double du nombre de coordinations dans une zone, il nous a semblé que cet indicateur devenait vraiment pertinant. D'une manière plus pratique, nous proposerons toujours des partitions de l'espace possédant une balance de partitionnement, soit proche de celle du découpage actuel, si celle-ci est inférieure à $2,0$, soit inférieure à $2,0$, dans le cas contraire.

Le choix d'une fonction de coût ne tenant pas compte de facteurs économiques (le ratio de coupe), signifie que nous cherchons à maximiser la sûreté et la fluidité du trafic. Mais le choix de tenir compte dans nos contraintes d'analyses économiques (les contraintes de tailles sont celles d'Eurocontrol, *cf. supra* 5.3.3) prouve que nous ne négligeons pas non plus cet aspect. Nous gardons cependant à

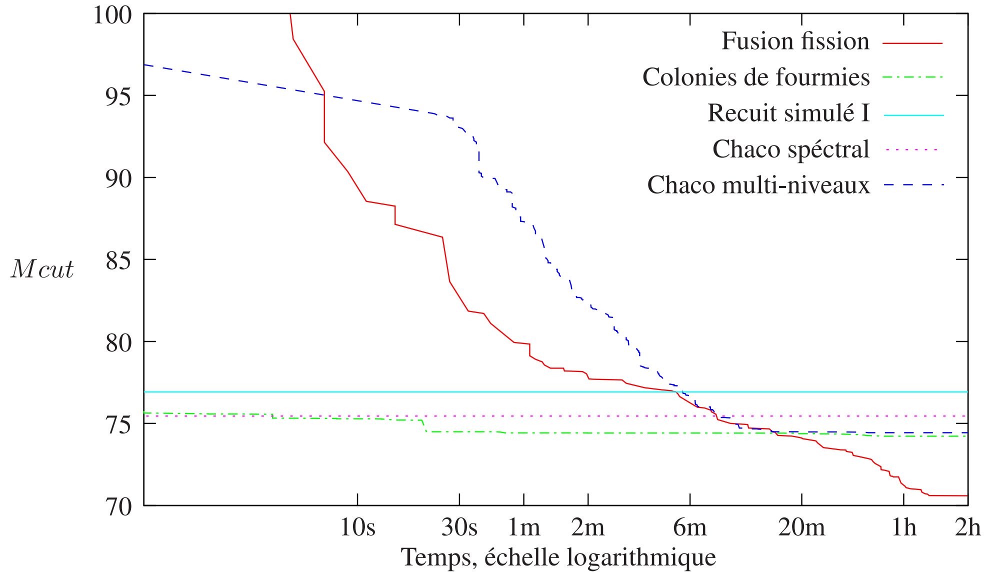


FIG. 5.5 – Comparaisons entre métaheuristiques, fusion-fission et Chaco.

l'esprit que, pour d'autres acteurs du monde de l'ATM[7], les objectifs et contraintes que nous venons de présenter sont différents. C'est pourquoi nos algorithmes ont été conçus pour pouvoir facilement s'adapter à d'autres fonctions objectifs et contraintes.

Maintenant que nous possédons un graphe des flux aériens en Europe et que les objectifs et contraintes de notre problème sont clairement définis, nous pouvons appliquer nos algorithmes à ce problème.

5.4 Résultats

Cette section présente et compare les résultats obtenus par les différents algorithmes de partitionnement de graphe non contraint que nous avons étudiés. Dans une première sous-section, les différentes métaheuristiques seront comparées entre elles (*infra* 5.4.1), puis la méthode de fusion-fission sera comparée avec des outils de partitionnement classiques (*infra* 5.4.2). Ces deux premières sous-sections permettront de trouver la méthode la plus performante pour notre problème. Nous pourrons alors présenter les découpages de l'espace aérien français (*infra* 5.4.3), du FAB d'Europe centrale (*infra* 5.4.4) et de la *core area* (*infra* 5.4.5), obtenus avec cette méthode, à savoir la fusion-fission.

Les résultats présentés dans cette section ont tous été obtenus avec un ordinateur fonctionnant avec le système d'exploitation GNU/Linux Debian, muni d'un processeur Intel Pentium IV cadencé à 3GHz, disposant de 512Mo de mémoire vive.

5.4.1 Comparaison avec les métaheuristiques

Cette sous-section a pour but de comparer les différentes applications des métaheuristiques que nous avons étudiées, sur le problème du découpage de l'espace aérien.

[7] ATM est l'acronyme de *Air traffic management*.

La figure 5.5 [BIC06a] présente une comparaison en fonction du temps, de la qualité des partitions trouvées par 5 méthodes : l'adaptation de la fusion-fission pour le partitionnement non contraint (*infra* 4.3), l'algorithme de colonies de fourmis (*supra* 3.3.2), la première adaptation du recuit simulé (*infra* 3.2.2), et les méthodes spectrales et multi-niveaux du logiciel Chaco (*supra* A.1). Les partitions qui ont été trouvées concernent le trafic de la journée du 8 février 2002 et concernent un découpage de la *core area* (*supra* 5.1.2) en 32 parties. Ce découpage n'a été créé que dans le but de comparer les méthodes entre elles. En effet, le logiciel Chaco ne permet que de trouver des partitions en 2^n parties (n entier).

Sur cette figure, les résultats du logiciel Chaco, qui ont pris chacun moins d'une seconde de temps de calcul, sont présents sous la forme de deux barres horizontales et représentent une forme de « palier » à partir duquel l'utilisation de métaheuristiques commence à prendre un sens. En plus des méthodes spectrales et multi-niveaux, le logiciel Chaco propose de nombreuses façons de partitionner un graphe. Nous avons testé de manière exhaustive toutes les façons possibles dans le cas des méthodes spectrales comme dans le cas de la méthode multi-niveaux, avant de sélectionner les deux plus performantes. La méthode spectrale qui a donné la meilleure partition utilise la méthode itérative du quotient de Rayleigh avec un k-partitionnement direct en octasection et la méthode d'affinage de Chaco. Et c'est la méthode spectrale de bissection directe avec affinage, qui a produit la meilleure partition pour le multi-niveaux.

Nous pouvons observer que l'algorithme de colonies de fourmis part avec un coût très petit. Ceci s'explique du fait qu'il est initialisé avec la méthode de percolation. De plus, afin d'optimiser le résultat obtenu avec la méthode de percolation, nous avons utilisé un recuit simulé pour optimiser le placement des sommets initiaux de la percolation. Le résultat obtenu est que la partition trouvée avec la méthode de percolation-recuit simulé, a un coût légèrement supérieur à celui de Chaco multi-niveaux. Et la méthode des colonies de fourmis trouve extrêmement vite une partition de coût inférieur à celui trouvé par Chaco multi-niveaux puis, après 10 secondes de calcul, à celui de Chaco spectral. Cependant, si la méthode des colonies de fourmis a besoin d'être initialisée avec une méthode comme la percolation (*supra* 3.3), ce n'est pas le cas du recuit, ni de la fusion-fission. Dans le cas du recuit, nous avons observé que l'algorithme mettait un certain temps à sortir de l'optimum local trouvé par la percolation et se déplaçait vers une région différente, ce qui rendait cette initialisation plus lente qu'une initialisation aléatoire et sans gain de performance. Nous avons aussi trouvé que la fusion-fission était beaucoup mieux initialisée par son propre processus d'initialisation (*supra* 4.3) que par un autre processus, qu'il soit aléatoire ou basé sur la percolation.

Nous pouvons observer à partir de la figure 5.5 que la méthode d'initialisation de la fusion-fission engendre une partition de coût élevé. Cependant, c'est à ce prix que se fait la descente assez rapide de la fusion-fission, qui passe successivement sous toutes les autres courbes. De même, l'initialisation aléatoire du recuit simulé le lance avec un peu de retard, cependant cela lui permet aussi d'avoir une forte progression et de trouver un partition de coût comparable avec celui des colonies de fourmis. Nous devons noter que les métaheuristiques atteignent un palier à partir de 40 minutes de calcul et la fusion-fission au bout d'1 heure 30. Cependant nous avons voulu voir si la fusion-fission franchissait ce palier, et une bonne heure après le début de celui-ci, l'algorithme a réussi à trouver d'autres solutions de coût plus bas.

Concernant le paramétrage des algorithmes, nous avons utilisé pour :

- le recuit simulé : $T_{max} = 2$, $T_{min} = 0$, $T_{seuil} = 1$, $min\_états\_refusés = 2$;
- les colonies de fourmis : 12 fourmis par colonie, $T = 0,01$, $t_{max} = 6$, $nb_f = 10$;
- la fusion-fission : $rfusion = rfission = \{0,85; 0,10; 0,04; 0,01\}$, $pas\_de\_règle = 0.001$, $coût\_min = 100$, $p = 0,5$, $q = 0,8$, $fission\_chaîne = 1,5$, $nombre\_de\_passes = 5$.

Le paramétrage de ces algorithmes c'est fait empiriquement. Comme nous l'avons précisé plus haut

Algorithmes	$Mcut$	Nb. secteurs max	Balance de partitionnement
Algorithme évolutionnaire	56,47	99	3,86
Colonies de fourmis	54,84	75	1,44
Fusion-fission	53,90	69	2,19

TAB. 5.2 – Comparaison entre algorithme évolutionnaire, algorithme de colonies de fourmis et fusion-fission pour un découpage de la *core area* en 24 parties.

(*supra* 4.3.9), les paramètres $rfusion$, $rfission$, $pas\_de\_règle$, $fission\_chaîne$, p et q de la fusion-fission ont été rapidement fixés, avant de faire varier plus finement $nombre\_de\_passes$ et $coût\_min$.

Dans le tableau 5.2, sont présentés les résultats de partitions obtenues avec la première adaptation de l'algorithme évolutionnaire au partitionnement non contraint (*supra* 3.4.2), ainsi qu'avec les colonies de fourmis et la fusion-fission. Les partitions trouvées concernent toujours le découpage de la *core area* européenne du 8 février 2002, mais en 24 parties.

Les résultats obtenus avec l'algorithme évolutionnaire sont, comme nous l'avions souligné lors de leur présentation, loin d'être satisfaisants, comparés à ceux des colonies de fourmis ou de la fusion-fission. Si le découpage trouvé par les colonies de fourmis a une très bonne balance, le nombre de secteurs de la zone la plus grande est bien trop grand. Cependant, le nombre de secteurs de la plus grande zone trouvée par la fusion-fission est lui aussi encore trop important, alors que la balance de partitionnement est un peu trop grande. Nous avons corrigé par la suite ces dérives de la fusion-fission. Cependant, sur cet exemple, c'est encore la fusion-fission qui trouve la partition de coût le plus faible, suivi des colonies de fourmis et de l'algorithme évolutionnaire.

Le même paramétrage des algorithmes que précédemment a été utilisé, avec cependant :
$coût\_min = 70$ et $nombre\_de\_passes = 10$.

La fonction de coût $Mcut$ utilisée pour ces comparaisons (*cf* figure 5.5 et tableau 5.2) est légèrement différente du ratio de coupe (*supra* 1.4). En utilisant les même notations qui ont servi à définir le ratio de coupe, alors :

$$Mcut(P_k) = \sum_{i=1}^{k} \frac{2 * coupe(S_i, S - S_i)}{2 * poids(S_i) - coupe(S_i, S - S_i)} .$$

Le facteur 2 devant $coupe(S_i, S - S_i)$ et $poids(S_i)$ s'explique par la nature particulière de notre problème : quand aéronef passe d'un secteur à un autre, il y a une charge de coordination pour chaque contrôleur des deux secteurs, c'est-à-dire deux charges de coordination. Cela explique le facteur 2 au numérateur. Au dénominateur, seul compte le nombre de charges de coordinations *dans* la zone de qualification, ainsi la coupe avec les zones extérieures sera compté une fois et non zéro, c'est pourquoi le facteur deux ne s'applique qu'à $poids(S_i)$. Nous avons gardé cette approche jusqu'au moment où nous nous sommes mis à comparer nos algorithmes avec plusieurs outils de partitionnement classiques. En effet, ceux-ci n'utilisant pas exactement les mêmes fonctions de coût que nous, nous nous sommes résolus à adopter précisément leurs fonctions pour rendre les comparaisons plus claires et plus vraies. Nous avons donc arrêté de multiplier le poids des arêtes par deux, car cela n'avait plus de sens dans le cas du partitionnement de graphe « classique ».

Ainsi, dans [BIC07], nous avons comparé les différentes méthodes avec une approche « classique », c'est-à-dire sans multiplier le poids des arêtes par deux.

Algorithmes	FF	SA	scotch	graclus	jostle	pmetis	kmetis
$ratio$	5,85	6,04	6,11	6,25	6,29	6,88	7,40
Nombre de sommets max	45,0	57,5	45,0	45,25	45,5	47,0	43,75
Balance	1,87	1,92	1,72	1,75	1,37	1,13	1,03
Temps de calcul	121s	122s	$\ll$1s	$\ll$1s	$\ll$1s	$\ll$1s	$\ll$1s
Distance avec FF	-	3,2%	4,3%	6,4%	7,0%	15,0%	20,9%
Écart type de $ratio$	0,331	0,250	0,184	0,233	0,337	0,123	0,260

TAB. 5.3 – Résultats moyens pour 100 permutations du graphe de trafic aérien de la journée du 17 juin 2005.

5.4.2 Comparaison avec des outils « classiques » de partitionnement

Après avoir constaté que la méthode de fusion-fission trouvait des partitions de coût un peu plus faible que celles trouvées par les trois autres métaheuristiques, nous avons voulu comparer de manière plus approfondie la fusion-fission et les outils « classiques » de partitionnement de graphe. Cette étude a été publiée dans [BIC07]. Elle compare les différents algorithmes sur la fonction objectif du ratio de coupe. Si dans [BIC07] cette fonction est celle de la coupe normalisée, c'est parce que nous considérions des sommets « fictifs » (c'est-à-dire n'intervenant pas dans la partition) possédant une arête avec d'autres secteurs pour représenter les flux de décollage et d'atterrissage des avions, ainsi que vers le reste du ciel aérien européen. Dans notre représentation actuelle du graphe, nous ne considérons plus de secteur fictif, ni les arêtes qui les liaient, mais avons ajouté le poids de ces arêtes dans le poids des sommets adjacents, comme nous l'avons expliqué (*supra* 5.3.5). Ainsi, la coupe normalisée utilisée sur notre ancienne approche du graphe de l'espace aérien européen est équivalente au ratio de coupe d'un graphe tel qu'il est présenté dans cette thèse.

Nous avons utilisé, dans cette nouvelle comparaison, la seconde adaptation du recuit simulé au partitionnement de graphe (*supra* 3.2.2). Comme nous l'avons fait remarquer dans la sous-section précédente (*supra* 5.4.1), la fusion-fission ne prend pas assez en compte la différence de taille entre les parties d'une partition. Afin de mieux prendre en compte la différence de taille entre les parties de la partition, nous avons utilisé la fonction objectif suivante :

$$f(P) = ratio(P) + \alpha * \left(\max_{P_i \in P} poids(P_i) - \min_{P_i \in P} poids(P_i) \right) ,$$

où la partition P vaut $P = \{P_1, \ldots, P_k\}$. Il s'agit donc de la même fonction objectif que pour le recuit simulé.

Les outils de partitionnement classiques comportent une partie aléatoire non négligeable. Ainsi, la qualité des partitions trouvées par ces outils est assez variable. Afin d'avoir une comparaison plus objective entre les différentes méthodes, les résultats présentés dans cette sous-section ont été obtenus pour 100 permutations des indices des sommets du graphe de trafic aérien de la journée du 17 juin 2005. Chaque permutation est en fait équivalente à changer la clef de la fonction de génération des nombres aléatoires des outils de partitionnement.

Pour avoir un éventail large des outils de partitionnement, nous en avons sélectionné 5 : pMetis et kMetis, Jostle, Scotch et Graclus. Ces quatre premiers outils cherchent à minimiser le coût de coupe quand le dernier minimise la coupe normalisée (*supra* A). Cependant, l'outil Scotch peut être en partie adapté à notre problème, car il possède une balance de partitionnement fortement adaptable. Il en est de même, mais dans une moindre mesure, pour l'outil Jostle. Enfin, comme dans le graphe de trafic

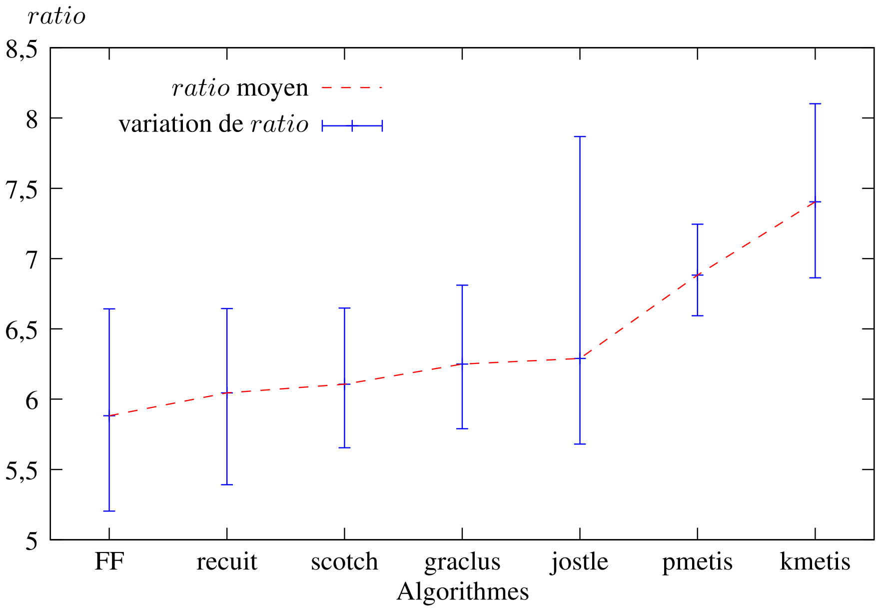


FIG. 5.6 – Écart et moyenne des ratios de coupe trouvés par les différents algorithmes pour 100 permutations du graphe d'origine.

aérien, la somme des poids des arêtes adjacentes à chaque sommet est proche de son poids, il est peu différent de minimiser la coupe normalisée et de minimiser le ratio de coupe.

Le tableau 5.4.2 compare la moyenne des résultats obtenus par les différentes méthodes pour 100 permutations du graphe de trafic aérien. La figure 5.6 présente la moyenne des ratios de coupe trouvés, ainsi que les valeurs maximales et minimales des ces valeurs. L'écart-type des ratios de coupe est inscrit en bas du tableau 5.4.2. La figure 5.7 résume valeurs moyennes et écarts des balances de partitionnement, ainsi que les valeurs moyennes et écarts des nombres de sommets maximaux des partitions trouvées. De ces différents résultats, nous proposons plusieurs observations concernant :

- **le ratio de coupe :** dans le tableau 5.4.2, les outils de partitionnement sont ordonnés par ordre croissant de leur valeur moyenne de ratio de coupe trouvé. La fusion-fission se trouve en première position, suivie du recuit simulé, puis des méthodes multi-niveaux avec, à leur tête, le logiciel Scotch. Ne vient qu'après le logiciel Graclus, alors que l'on aurait pu s'attendre à ce qu'il soit l'un des premiers, au vu de sa fonction objectif. Ce dernier est au coude à coude avec le logiciel Jostle, puis apparaissent les deux outils de Metis en queue de peloton. La figure 5.6 confirme ce classement avec une surprise : la volatilité de la qualité des partitions trouvées par Jostle. Quant à la volatilité de la qualité des partitions trouvées par la fusion-fission et le recuit simulé, elle vient en partie du fait que ces deux algorithmes ont été stoppés tôt dans leur processus d'optimisation (2 minutes) ;
- **le nombre maximal de secteur :** tous les algorithmes trouvent des partitions qui, en moyenne, respectent cet écart type, sauf le recuit simulé qui en est loin. Cependant, en durcissant le paramètre α plus que ce que nous avons fait, les solutions sont extrêmement mauvaises. La figure 5.7 nous montre que toutes les méthodes ont trouvé au moins une partition dont le nombre de secteurs maximal était supérieur à 50, à l'exception de la fusion-fission qui atteint sans la dépasser cette limite ;
- **la balance de partitionnement :** les moyennes des balances décrites dans le tableau 5.4.2 res-

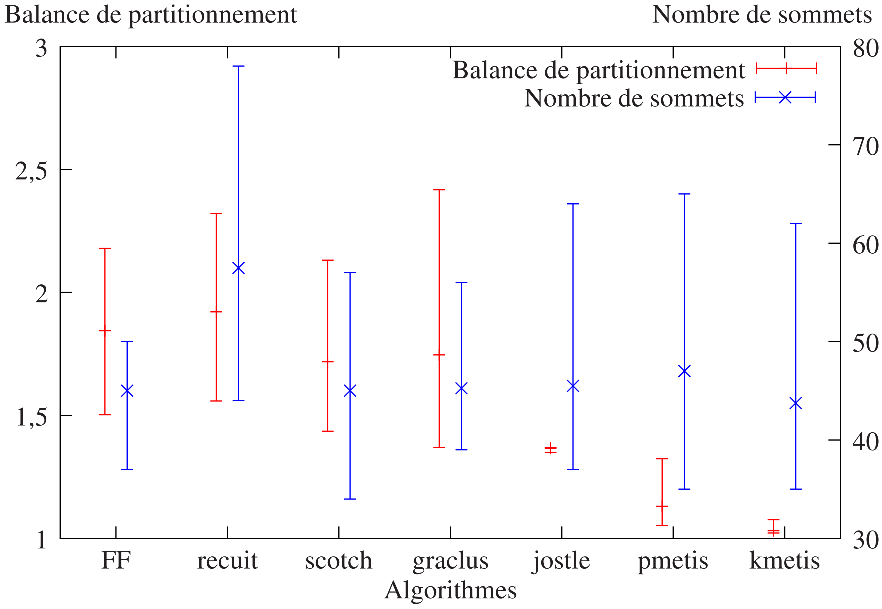


FIG. 5.7 – Écart et moyenne des balances de partitionnement et des nombres de sommets maximaux trouvés par les différents algorithmes pour 100 permutations du graphe d'origine.

pectent toutes la balance de partitionnement de $2,0$. Cependant, dans quelques cas, les méthodes de fusion-fission, du recuit simulé, Scotch et Graclus trouvent des partitions ne respectant pas cette balance (*cf* figure 5.7). Il faut faire remarquer que se sont les mêmes outils qui permettent de trouver les ratios de coupe les plus petits et les balances les plus grandes. Cela permet de confirmer que ces deux caractéristiques sont bien liées comme nous l'avions dit plus tôt (*supra* 1.5) ;

– **le temps de calcul :** c'est sans surprise que l'on constate que le recuit et la fusion-fission ont un temps de calcul beaucoup plus élevés que celui des outils classiques. D'autant plus que ces outils ont été largement optimisés au niveau de leur rapidité. Le temps maximal d'exécution de ces deux algorithmes a été limité dans le but de pouvoir les lancer une centaine de fois, mais aussi afin d'avoir des temps d'exécution encore « rapides ». De plus, nous avons constaté qu'en programmant de façon basique les algorithmes de pMetis [KAR98a], on obtenait des temps de calcul de l'ordre de quelques minutes, bien loin de ceux de ces outils très optimisés et bien paramétrés. Pour donner un exemple, quand l'algorithme de Kernighan-Lin n'est pas bien paramétré, le nombre de passes peut être inutilement élevé, ainsi que le nombre d'exécutions de cet algorithme.

L'algorithme de fusion-fission a la particularité de pouvoir trouver plusieurs partitions de bonne qualité de nombres de parties différents en une seule exécution. La figure 5.8 compare l'exécution d'une instance de l'algorithme de fusion-fission pour $k = 26$, avec 13 instances de chaque outil Scotch ou Graclus, pour $k \in \{18, \dots, 31\}$. Les courbes représentent le pourcentage du ratio de coupe de chaque outil en fonction du ratio de coupe trouvé par Graclus. Pour 7 nombres de parties, l'algorithme de fusion-fission trouve des partitions dont le ratio de coupe est plus petit que celui du logiciel Scotch (pour $k \in \{22, \dots, 28\}$). Et pour 9 nombres de parties, l'algorithme de fusion-fission trouve des partitions dont le ratio de coupe est plus petit que celui du logiciel Graclus (pour $k \in \{21, \dots, 29\}$).

Paramétrage des algorithmes :

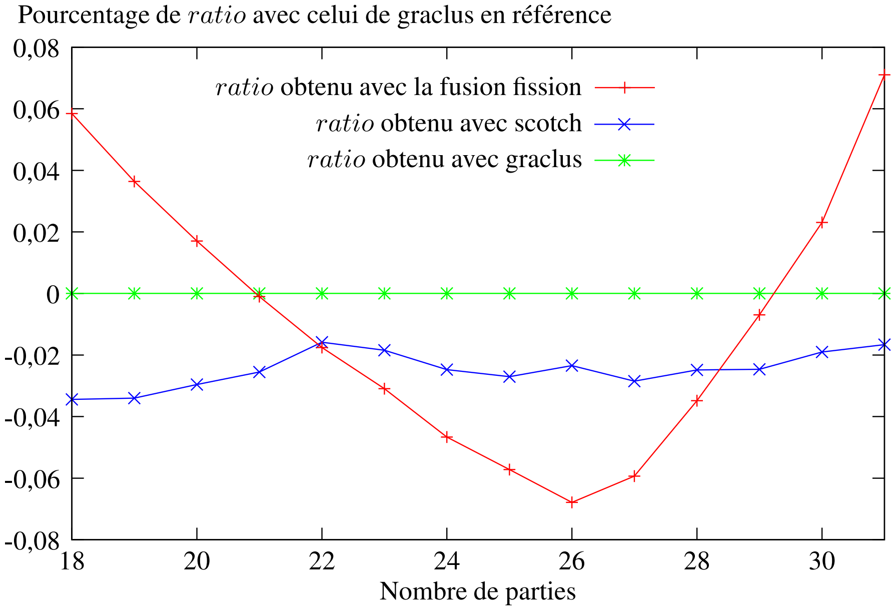


FIG. 5.8 – Comparaison entre les résultats : d'une exécution de l'algorithme de fusion-fission pour $k = 26$, de 13 exécutions de l'outil Scotch pour $k \in \{18, \ldots, 31\}$, et de 13 exécutions de l'outil Graclus pour $k \in \{18, \ldots, 31\}$. Le ratio de coupe des partitions trouvées par ces différentes méthodes est présenté en pourcentage de celui trouvé par l'outil Graclus.

- fusion-fission : $rfusion = \{0,5; 0,2; 0,2; 0,07; 0,03; 0,01\}$, $rfission = \{0,8; 0,15; 0,05; 0,0\}$, $pnucl\_éjectés = \{0,0; 0,01; 0,02; 0,03; 0,04; 0,05\}$, $pas\_de\_règle = 0.001$, $fission\_chaîne = 1,7$, $p = 0,2$, $q = 0,3$, $coût\_min = 14$, $nombre\_de\_passes = 42$, $\alpha = 76 * 10^{-6}$;
- recuit simulé : $T_{max} = 0,011$, $T_{fact} = 0,95$, $mult\_taille = 15$, $min\_pourcent = 0,01$, $\alpha = 16 * 10^{-6}$;
- Scotch : avec les notations du manuel [PEL06], nous utilisons la stratégie suivante : $MULTI = m\{asc = FM, low = h\{pass = 10\}FM, type = h, vert = 80, rat = 0.7\}$ avec $FM = f\{move = 80, pass = -1, bal = 0.1\}$. Ce qui correspond à l'utilisation d'une méthode multi-niveaux avec pour méthode d'affinage celle de Fiduccia-Mattheyses. La balance de partitionnement recherchée est $1,2$, ce qui correspond dans la librairie Scotch à $bal = 0.1$;
- Graclus : un niveau de recherche locale de 4 a été choisi ;
- Jostle : nous avons utilisé comme balance $1,37$, ce qui correspond après une recherche exhaustive au ratio de coupe le plus bas ;
- Metis : aucun paramétrage particulier n'a été fait pour pMetis et kMetis.

Les valeurs des paramètres de la fusion-fission et du recuit simulé ont été obtenus de manière empirique. Dans le cas de la fusion-fission, nous avons procédé comme décrit à la sous-section 4.3.9. Cependant, aux deux paramètres $nombre\_de\_passes$ et $coût\_min$ à régler précisément, vient s'ajouter un paramètre particulier spécifique au découpage aérien, le paramètre α. Les paramètres des outils de partitionnement ont été choisis en tenant compte de la documentation de ces outils.

Les différents résultats que nous venons de voir montrent que la méthode de fusion-fission est la plus efficace pour trouver un redécoupage de l'espace aérien européen. C'est pourquoi, dans les sections suivantes, nous présenterons toujours des résultats obtenus avec cette méthode.

Les essais qui viennent d'être présentés ont servi à paramétrer les algorithmes. Dans les sections suivantes, nous emploierons la méthode de fusion-fission et comparerons ses résultats avec ceux des outils Scotch et Graclus. Si l'outil Graclus ne nécessite pour tout paramètre que le graphe et le nombre de parties de la partition, l'algorithme de fusion-fission en utilise beaucoup, même si comme nous l'avons vu (*supra* 4.3.9), il n'est pas nécessaire de régler précisément la plupart de ces paramètres. Les paramètres suivant de l'algorithme de fusion-fission ont été utilisés dans tous les tests relatifs au découpage de l'espace aérien qui vont suivre :

- $rfusion = \{0,5; 0,2; 0,2; 0,07; 0,03; 0,01\}$;
- $rfission = \{0,8; 0,15; 0,05; 0,0\}$;
- $pnucl\_éjectés = \{0,0; 0,01; 0,02; 0,03; 0,04; 0,05\}$;
- $pas\_de\_règle = 0.001$;
- $p = 0,2$, $q = 0,3$;
- $fission\_chaîne = 1,7$.

Ces paramètres fixés, il en reste 3 qui vont pouvoir être ajustés :

- α : il est essentiel, car il permet de régler la balance de partitionnement ;
- $coût\_min$: de lui dépend la fin de l'étape d'initialisation ;
- $nombre\_de\_passes$.

Le paramètre $coût\_min$ n'est pas un paramètre qui se règle finement. Nous l'utilisons de telle manière que l'algorithme d'initialisation ait un temps d'exécution le plus court possible, à ce titre la valeur de $coût\_min$ est toujours beaucoup plus grande que la valeur du ratio de coupe de la partition finale trouvée. Enfin, le paramètre $nombre\_de\_passes$ peut être ajusté ; cependant, la valeur de 42 présentée dans cette sous-section donne en général de bonnes partitions.

Nous avons gardé par la suite le même paramétrage de l'outil Scotch que celui utilisé dans cette sous-section, c'est-à-dire :

$$MULTI = m\{asc = FM, low = h\{pass = 10\}FM, type = h, vert = 80, rat = 0.7\} ,$$

avec

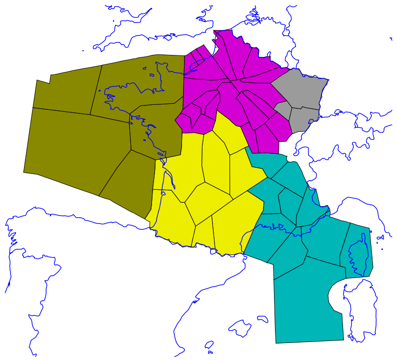
$$FM = f\{move = 80, pass = -1, bal = bal\} .$$

Le seul paramètre qui sera changé va être la balance de partitionnement bal.

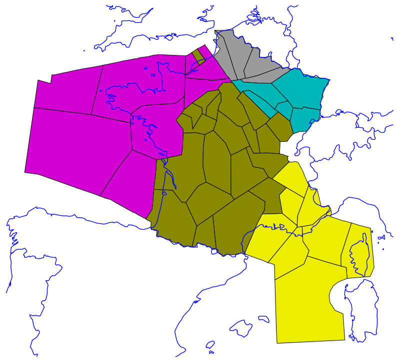
Dans les sous-sections qui vont suivre, les résultats proposés seront à chaque fois le fruit de plusieurs tests pour la fusion-fission comme pour l'outil Scotch. Pour le paramétrage de la fusion-fission, nous conseillons de commencer par régler le paramètre α influençant la balance de partitionnement, puis d'ajuster $coût\_min$ et en dernier, de modifier $nombre\_de\_passes$. Concernant le paramètre α, il faut éviter de lui donner une valeur trop grande pour obtenir une balance plus faible. En effet, plus le temps d'exécution de notre algorithme de notre algorithme est élevé, plus la partition trouvée a une petite balance. Dans une certaine mesure, nous pouvons dire que notre algorithme procède en deux temps, pendant le premier temps, il explore l'espace de recherche jusqu'à trouver un large « puits » de solutions, qu'il explore dans un second temps, en diminuant en parallèle la balance de partitionnement et le ratio de coupe. Or, dans la première étape, si le paramètre α est plus permissif, alors l'exploration aura plus de chances d'arriver dans une région de solutions de bonnes qualités.

5.4.3 Découpage de l'espace aérien français

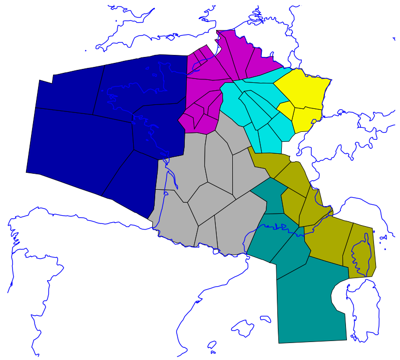
Le premier problème de découpage que nous avons étudié concerne le ciel aérien français. En comparaison avec les autres pays européens (*cf* tableau 5.1), on constate que les centres français possèdent des tailles assez proches des recommandations d'Eurocontrol. La France est découpée en



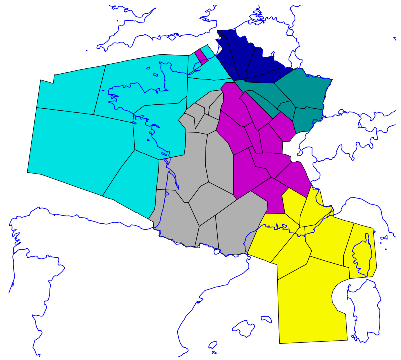
(a) Centres de contrôle (5) au FL200



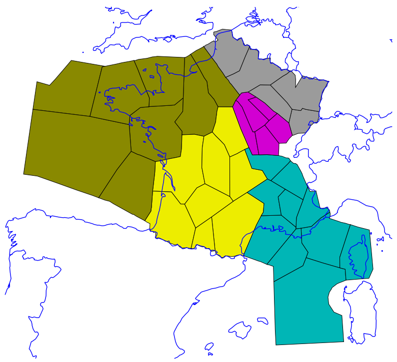
(b) Découpage en 5 parties trouvé par la fusion-fission au FL200



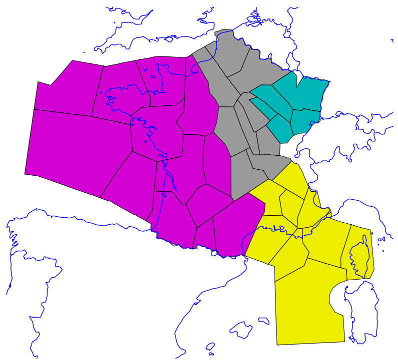
(c) Zones de qualification (7) au FL200



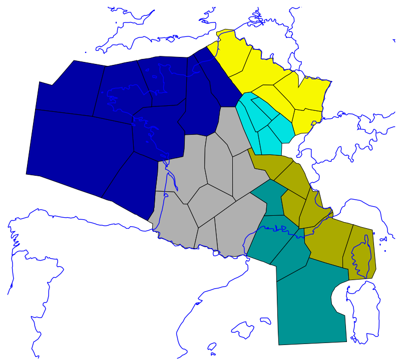
(d) Découpage en 7 parties trouvé par la fusion-fission au FL200



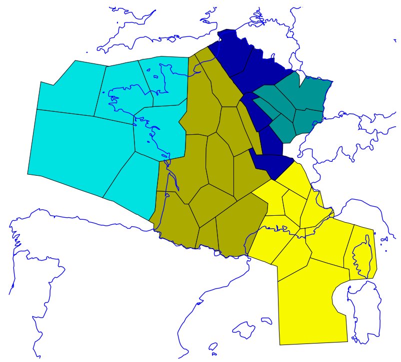
(e) Centres de contrôle (5) au FL400



(f) Découpage en 5 parties trouvé par la fusion-fission au FL400



(g) Zones de qualification (7) au FL400



(h) Découpage en 7 parties trouvé par la fusion-fission au FL400

FIG. 5.9 – Découpage de l'espace aérien français : en centres de contrôle (figures 5.9(a) et 5.9(e)) et en zones de qualification (figures 5.9(c) et 5.9(g)). Résultats trouvés par l'algorithme de fusion-fission pour un partitionnement en 5 parties (figures 5.9(b) et 5.9(f)) et 7 parties (figures 5.9(d) et 5.9(d)).

Algorithmes	*ratio*	*coupe*	Balance	Max secteurs	Min secteurs
Centres, le 17/6/2005	0,870	6 442	1,15	38	13
Fusion-fission	0,579	4 472	1,37	39	12
Scotch	0,711	5 003	1,41	33	23
Graclus	0,611	4 759	1,47	45	12

TAB. 5.4 – Résultats du découpage de l'espace aérien français en 5 parties.

139 secteurs de contrôle en route. Si l'on suit précisément les recommandations d'Eurocontrol, et en considérant qu'un nombre de secteurs par centre un peu plus grand est préférable en terme de coût financier, à un nombre plus petit (*cf* figure 5.4), alors la France devrait comporter 5 centres de contrôle en route avec un zone de qualification dans chaque centre. Comme nous l'avons vu, la France possède déjà 5 centres de contrôle en route, mais 7 zones de qualification. Nous proposerons donc dans cette sous-section une étude des découpages en 5 et 7 parties, *i.e.* zones de qualification, de l'espace aérien français.

Les résultats présentés dans cette sous-section ont été réalisés à partir des plans de vols de la journée du 17 juin 2005.

Quelques caractéristiques du découpage du ciel aérien français en centres de contrôle sont énumérées au début du tableau 5.4. La suite de ce tableau regroupe les résultats du partitionnement effectué par la fusion-fission, et les outils Scotch et Graclus. C'est la fusion-fission qui permet de trouver la partition à la fois de ratio de coupe, de coût de coupe et de balance la plus faible. Le nombre maximum de secteurs trouvé par Scotch est plus petit, cependant tous les résultats trouvés respectent pleinement les recommandations d'Eurocontrol. En considérant le ratio de coupe et le coût de coupe, on observe que le découpage actuel en centres de contrôle est relativement bon, bien que pouvant être amélioré, au vu des critères que nous avons établi pour le découpage en zones de qualification. Cependant, en comparant les coûts de coupe entre le découpage existant et celui trouvé par la fusion-fission, alors on s'aperçoit que près de 2 000 passages d'avions entre centres pourraient être évités en une journée. L'analyse du nombre de secteurs minimum des parties de la partition permet de connaître la taille minimale que pourrait avoir une zone de qualification. Comme nous l'avons vu, si un centre de petite taille est proportionnellement plus coûteux qu'un centre plus grand (*supra* 5.3.3), cela est moins vrai pour les zones de qualification, puisque plusieurs d'entre elles peuvent être regroupées dans un même centre.

Dans la mesure où la France compte actuellement 7 zones de qualification et afin de présenter une comparaison objective des résultats obtenus avec nos algorithmes, nous présentons aussi un découpage en 7 parties du ciel aérien français. Le tableau 5.5 présente quelques caractéristiques du découpage réel, et les résultats obtenus par la méthode de fusion-fission et les outils Scotch et Graclus. Le découpage réel en zones de qualification a une balance plus élevée que celle du découpage en centres, ce qui pourrait indiquer que les centres sont plus équilibrés en terme de charge de trafic que les zones de qualification en France. La partition en 7 parties trouvée par la fusion-fission est la plus performante sur les 4 critères proposés. Celle-ci a une balance très faible, ce qui permet d'avoir des zones dont la difficulté de contrôle est « semblable ». Ces zones de qualification comportent au plus 26 secteurs, ce qui est au dessous de la recommandation de 28 secteurs que nous avait proposée quelques contrôleurs.

La figure 5.9 présente en parallèle une série de découpages de l'espace aérien français vue en coupe aux niveaux de vol FL200 et FL400. Le découpage réel français en 5 centres de contrôle (figures 5.9(a) et 5.9(e)) se situe tout à gauche. Il est suivi de la partition trouvée par la fusion-fission en 5

Algorithmes	*ratio*	*coupe*	Balance	Max secteurs	Min secteurs
Zones, le 17/6/2005	1,65	8 419	1,53	31	13
Fusion-fission	1,08	5 853	1,15	26	12
Scotch	1,14	6 030	1,66	37	12
Graclus	1,47	7 576	1,40	28	10

TAB. 5.5 – Résultats du découpage de l'espace aérien français en 7 parties.

parties (figures 5.9(b) et 5.9(f)). La représentation de zones de qualification réelles lui succède (figures 5.9(c) et 5.9(g)). Enfin, le découpage en 7 parties trouvée par la fusion-fission termine cette succession (figures 5.9(d) et 5.9(h)).

Ces figures ont été présentées à la file afin de pouvoir constater les similitudes entre le découpage trouvé par les fusion-fission en 7 parties et celui en 5 parties. En effet, au FL200, les deux parties du centre de la France sur la figure 5.9(c) ne font plus qu'une partie sur la figure 5.9(a). Et au FL400, les parties du centre et de l'est (figure 5.9(g)) sont elles aussi regroupées en une partie unique (figure 5.9(e)). Dans le premier cas, seul un secteur diffère, dans le second, quatre. Ce constat inattendu, renforcé par la balance et le nombre maximal de secteurs trouvés par la fusion-fission, conforte les paramètres actuels du découpage français en 5 centres de contrôle et 7 zones de qualification.

Plusieurs remarques apparaissent concernant les découpages trouvés par la fusion-fission en comparaison de ceux existant :

- la partie s'occupant des arrivées venant de l'Atlantique ne fait toujours qu'un bloc, sur tous les niveaux de vol ;
- la traversée nord sud de la France aux niveaux supérieurs (à partir de FL285) a une forme bien différente des niveaux inférieurs (FL100-285) ;
- la région sud-est est délesté des plusieurs secteurs et ainsi, n'occupe plus qu'une seule partie, au lieu de deux zones de qualification ;
- l'approche de Paris est très marquée dans les niveaux inférieurs, et prend la forme d'une « étoile », mais l'est beaucoup moins dans les niveaux supérieurs, ce qui correspond au survol de Paris à très haute altitude ;
- le nord-est, très dense en terme de trafic est coupée en deux, une partie orientée vers l'Allemagne et l'autre vers la Belgique. Cette dernière est reliée au sud de la France dans les niveaux supérieurs, par un « couloir » aérien.

5.4.4 Découpage de l'espace aérien du bloc fonctionnel d'Europe centrale

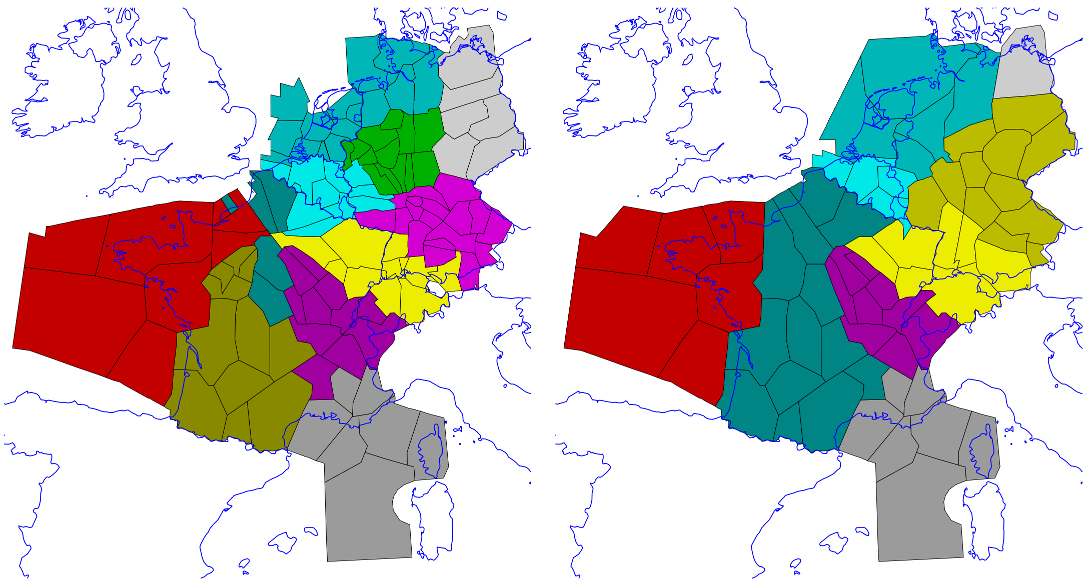
Le projet de découpage de l'Europe centrale consiste à rassembler 6 pays européens (*supra* 5.2.2) : l'Allemagne, la Belgique, la France, le Luxembourg, les Pays-Bas et la Suisse. Ces six pays possèdent actuellement 16 centres de contrôle en route et 26 zones de qualification, pour un total de 300 secteurs. En appliquant les recommandations d'Eurocontrol, le FAB d'Europe centrale devrait posséder 12 centres de contrôle. Nous proposerons dans cette sous-section trois découpages de l'espac aérien du bloc fonctionnale d'Europe centrale, le premier en 12 parties, le second en 16 parties et le troisième en 26 parties, chacun d'eux pouvant représenter un découpage en zones de qualification du FAB d'Europe centrale.

Le tableau 5.6 récapitule les valeurs caractéristiques des découpages existants ou créés de l'espace aérien du FAB d'Europe centrale. Ce tableau se divise en trois parties :

- la première partie de ce tableau propose plusieurs découpages en 12 parties. Le résultat trouvé par la fusion-fission possède le ratio de coupe et le coût de coupe le plus intéressant, en étant

Algorithmes	*ratio*	*coupe*	Balance	Max & min secteurs	
Découpage en 12 parties					
Fusion-fission	2,25	18 633	1,27	35	17
Scotch	2,43	19 231	1,26	32	16
Graclus	2,68	21 288	1,38	38	14
Paramètres	Scotch : $bal = 0,05$ Fusion-fission : $co\hat{u}t\_min = 14$, $nombre\_de\_passes = 12$, $\alpha = 8 * 10^{-5}$				
Découpage en 16 parties					
Centres, le 17/6/2005	4,36	24 875	1,91	38	7
Fusion-fission	3,55	21 674	1,28	26	15
Scotch	3,82	23 094	1,20	29	14
Graclus	4,03	24 524	1,42	28	12
Paramètres	Scotch : $bal = 0,05$ Fusion-fission : $co\hat{u}t\_min = 12$, $nombre\_de\_passes = 56$, $\alpha = 10^{-4}$				
Découpage en 26 parties					
Zones, le 17/6/2005	9,35	29 669	3,11	31	3
Fusion-fission	6,97	26 393	1,55	20	6
Scotch	7,44	27 785	1,64	18	7
Graclus	8,00	30 152	1,75	18	4
Paramètres	Scotch : $bal = 0,12$ Fusion-fission : $co\hat{u}t\_min = 12$, $nombre\_de\_passes = 56$, $\alpha = 10^{-4}$				

TAB. 5.6 – Résultats du découpage de l'espace aérien du FAB d'Europe centrale en 12, 16, et 26 parties



(a) Découpage en 12 parties trouvé par la fusion-fission au FL200.

(b) Découpage en 12 parties trouvé par la fusion-fission au FL400.

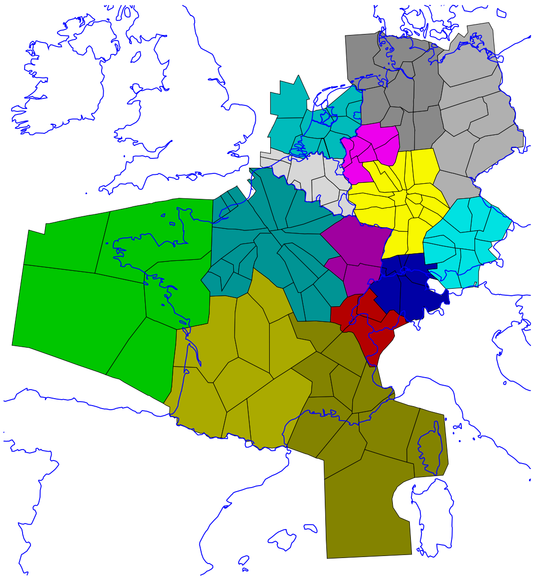
FIG. 5.10 – Découpage en 12 parties de l'espace aérien du FAB d'Europe centrale, obtenu par l'algorithme de fusion-fission.

à égalité avec l'outil Scotch pour la balance de partitionnement. Ce dernier a un nombre maximum de secteurs un peu moins élevé, mais ce nombre est bien inférieur dans les trois cas aux limites proposées par Eurocontrol. Enfin, avec un nombre minimal de 17 secteurs, la fusion-fission donne là aussi le résultat le plus intéressant. La partition la plus intéressante semble donc être celle de la fusion-fission ;

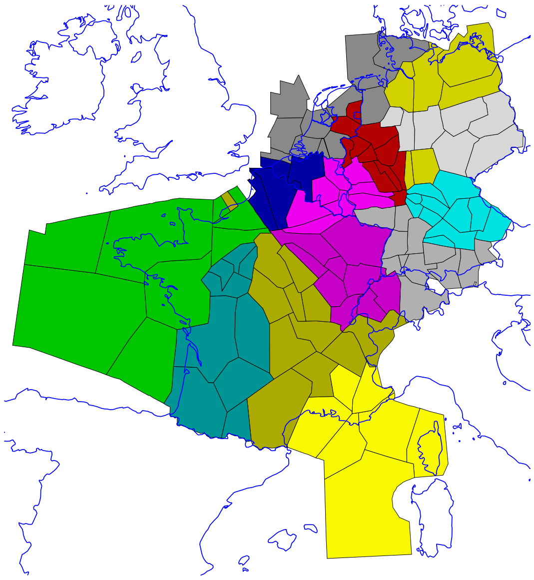
- la seconde partie de ce tableau compare le découpage réel en 16 centres de contrôle du 17 juin 2005, aux partitions trouvées par les trois autres méthodes en 16 parties. Les coûts et ratio de coupe du découpage réel sont très proches de ceux de la partition trouvée par l'outil Graclus. Ainsi, une première impression pourrait laisser penser que le découpage réel nécessite peu d'améliorations. Cependant, la balance de la partition de Graclus est bien inférieure à celle du découpage réelle. Or, la diminution de la balance d'une partition va de pair avec l'augmentation de son ratio de coupe. Concernant le reste des résultats, c'est la partition trouvée par la fusion-fission qui est la plus performante, avec cependant une balance de partitionnement légèrement supérieure à celle de la partition trouvée par Scotch ;
- la troisième et dernière partie de ce tableau présente les caractéristiques du découpage réel en 26 zones de qualification et des partitions trouvées par les trois méthodes de partitionnement en 26 parties. La balance de partitionnement du découpage réel est si élevée que l'on peut supposer que les zones de qualification de ces 6 pays sont de difficultés de contrôle inégales. Dans ce cas encore, c'est l'algorithme de fusion-fission qui permet de trouver la partition de ratio, de coût et de balance les plus faibles. Cependant, les nombres de secteurs maximaux et minimaux sont pour toutes les partitions trouvées très petits comparés aux recommandations que nous avons eues des contrôleurs ; ceci est dû au très grand nombre de zones de qualification actuel de ces 6 pays.

La figure 5.10 illustre le découpage trouvé par la méthode de fusion-fission de l'espace aérien du FAB d'Europe centrale en 12 parties. En mettant côte à côte ces résultats avec le découpage actuel en 12 centres de contrôle (figure 5.11(a)), plusieurs similitudes apparaissent : aux niveaux de vol inférieurs, les parties de la partition situées à l'ouest, au sud-ouest et au sud de la France se retrouvent dans les deux figures, ainsi que la partie en Allemagne de l'est ; toujours au niveau de vol moyen, certaines parties de la partition trouvée avec la fusion-fission semblent être le regroupement de deux centres existants : Pays-Bas, Allemagne du nord-ouest, et Suisse de l'est, France de l'est ; aux niveaux de vol supérieurs (après FL285), les correspondances entre centres et parties sont plus nettes au-dessus de l'Allemagne et du Bénélux, avec pour le centre du Bénélux (qui est en fait celui d'Eurocontrol) une division en deux partie dans le découpage de la fusion-fission. On peut noter aux niveaux supérieurs que le découpage de la fusion-fission fait apparaître en Belgique une partie très marquée sur ses frontières, alors même que ce pays se trouve en plein milieu du découpage. Le découpage que nous étudions est basé sur des secteurs qui ont eux-mêmes été créés sur la base de frontières nationales et ainsi, tout redécoupage basé sur ces secteur sera obligatoirement biaisé par la forme de ces secteurs. De même, ce redécoupage est basé sur les *routes aériennes* existantes. Pour être le plus performant possible, un redécoupage nécessiterait la modification de ces *routes aériennes*, ainsi que de la géographie de ces secteurs.

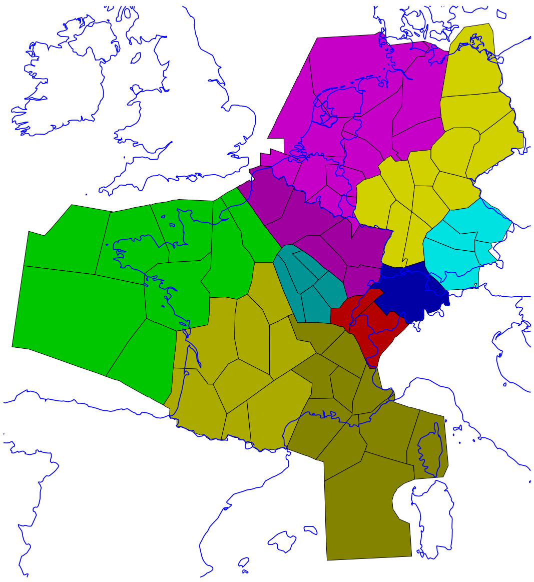
Si nous revenons au découpage de l'espace aérien français en 26 parties trouvé par la méthode de fusion-fission (figures 5.9(d) et 5.9(h)), alors on constate que celui-ci est très similaire, pour la France, au découpage en 12 parties du FAB d'Europe centrale trouvé par la même méthode (figures 5.10(a) et 5.10(b)). Aux niveaux de vol inférieurs, la seule partie qui diffère est celle du nord qui est divisé en deux et déborde vers le sud, sur la figure 5.10(a). Aux niveaux de vol supérieurs, cette même partie du nord de la France sur la figure 5.9(h) est divisée en deux dans la figure 5.10(b), une partie allant vers le centre Suisse-est et l'autre vers le centre France-centre.



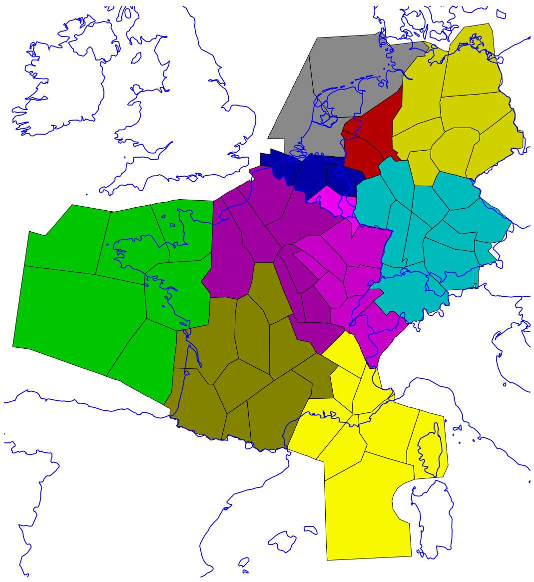
(a) Centres de contrôle (16) au FL200.



(b) Découpage en 16 parties trouvé par la fusion-fission au FL200.

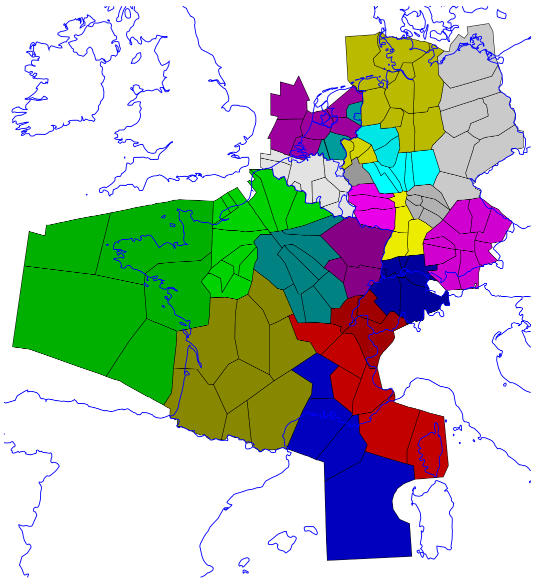


(c) Centres de contrôle au FL400.

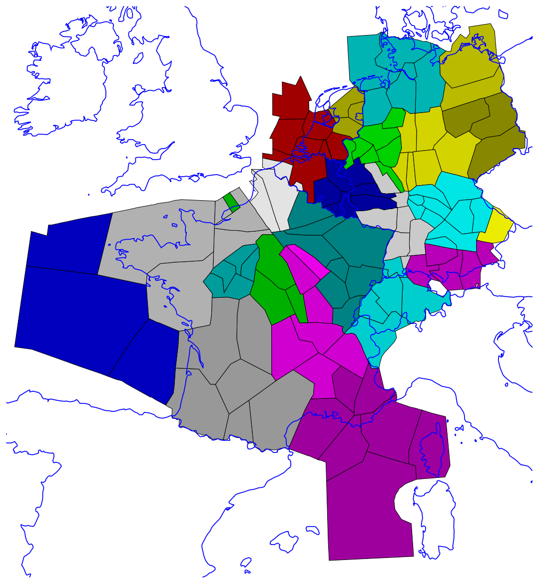


(d) Découpage en 16 parties trouvé par la fusion-fission au FL400.

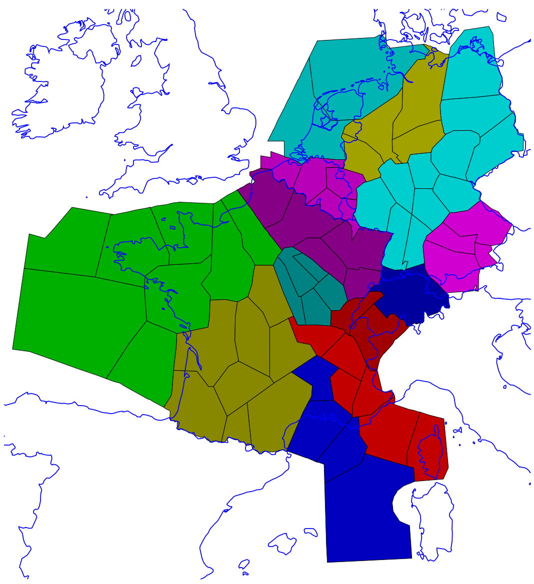
FIG. 5.11 – Représentation des 16 centres de contrôle de l'espace aérien du FAB d'Europe centrale, et découpage en 16 parties calculé par la fusion-fission.



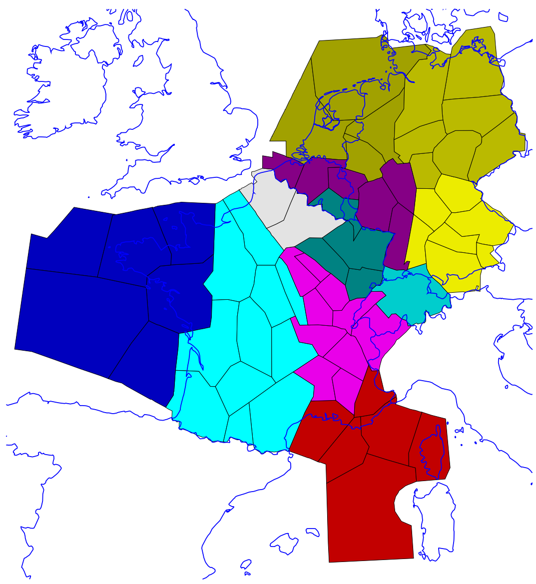
(a) Zones de qualification (26) au FL200.



(b) Découpage en 26 parties trouvé par la fusion-fission au FL200.



(c) Zones de qualification (26) au FL400.



(d) Découpage en 26 parties trouvé par la fusion-fission au FL400.

FIG. 5.12 – Représentation des 26 zones de qualification du FAB d'Europe centrale, et découpage en 26 parties calculé par la fusion-fission.

La figure 5.11 rassemble le découpage réel en 16 centres de contrôle du FAB d'Europe centrale avec la partition en 16 parties trouvée par la fusion-fission sur le même espace aérien. Aux niveaux inférieurs, les similitudes des découpages trouvés par la fusion-fission son fortes entre le découpage trouvé pour la France (figure 5.9(d)) et celui trouvé pour le FAB d'Europe centrale, en observant que la France (figure 5.11(b)). Dans ces deux résultats, Paris est au centre d'une étoile formée par dans un cas par les centres de contrôle français, dans l'autre par les parties françaises de la partition. La ressemblance est toujours très forte dans le cas des niveaux de vol supérieurs (figures 5.9(h) et 5.11(d)). Cette observation est un point positif pour la fusion-fission, qui permet de trouver des solutions homogènes, alors quand la complexité augmente.

Par contre, si l'on compare le découpage réel en centres de contrôle du FAB d'Europe centrale et la partition en 16 parties trouvé par la fusion-fission, alors, on constate que ces deux découpage sont bien différents. Les plus grandes ressemblances que nous pouvons observer concernent les parties France-ouest et France-Sud, pour tous les niveaux de vol. Il est intéressant de constater qu'aux niveaux de vol inférieurs, les parties allemandes trouvées par la fusion-fission semblent entourer Francfort comme c'est le cas pour Paris en France. Plusieurs « axes » apparaissent dans ce découpage : Francfort-Paris, Bruxelles-Paris, Genève-Paris, Francfort-Zurich et Francfort-Munich. Si ces axes ressortent du découpage aux niveaux de vol inférieurs, ce n'est plus le cas aux niveaux de vol supérieurs. On peut aussi remarquer que dans les niveaux de vol supérieurs, le découpage suit plus les frontières étatiques que dans les niveaux de vol inférieurs.

La figure 5.12 présente le découpage réel des pays du FAB d'Europe centrale en zones de qualification. Avec 26 zones de qualification, la moyenne du nombre de secteurs par zones qui vaut 12, est très faible. Il n'est donc pas étonnant que le découpage trouvé par la fusion-fission (figures 5.12(b) et 5.12(d)) contienne un grand nombre de parties différentes. Cependant, la constatation qui s'impose dans cette partition est que le nombre de parties présentes aux niveaux inférieurs est bien plus important que celui des niveaux supérieurs. Cela correspond à la nature des trafics aux différentes altitudes de vol. En effet, les niveaux de vol inférieurs sont au contact des régions d'approches des grands aéroports et leur but est de faire monter ou descendre les avions. Dans ces niveaux de vol, les parties vont donc avoir tendance à être très hautes, mais à avoir une base de superficie réduite. Alors que dans le cas des niveaux de vol supérieurs, les avions sont tous en altitude de croisière, ce qui va avoir pour répercussion d'augmenter les superficies des parties, mais aussi de les aplatir. Ceci explique pourquoi nous allons trouver plus facilement des parties structurées autour d'axes de trafic aux abords ou entre grands aéroports, dans les niveaux de vols inférieurs, et des parties plus étendues aux niveaux de vol supérieurs.

5.4.5 Découpage de l'espace aérien de la *core area* européenne

Nous avons vu (*supra* 5.1.2) que la *core area* est la région de plus forte densité de trafic du ciel européen. Dans l'étude réalisée pour la définir [Bic05], elle est composée des 11 pays suivants : l'Espagne, l'Italie, la Suisse, l'Autriche, l'Allemagne, la Belgique, le Luxembourg, les Pays-Bas, le Danemark, la France et le Royaume-Uni. Dans cette section, nous nous intéresserons au redécoupage de la *core area*. Il nous a aussi été demandé d'ajouter à cette étude l'Irlande. Ainsi, nous désignerons par *core area* dans cette section, en plus des pays qui la composent, l'Irlande.

La *core area* était constituée en juin 2005 de 31 centres de contrôle et de 50 zones de qualification pour un total de 638 secteurs (*supra* 5.1.2). En suivant les recommandations d'Eurocontrol concernant la tailles des centres de contrôle, alors ceux-ci devraient être au nombre de 25. Nous proposons dans cette sous-section des découpages de la *core area* en 25, 31 et 50 parties, chacun de ces découpage pouvant être assimilé à un découpage en autant de zones de qualifications.

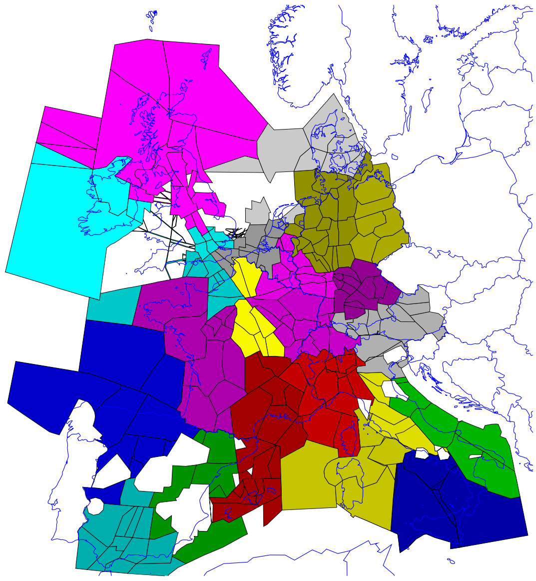
Algorithmes	$ratio$	$coupe$	Balance	Max & min secteurs	
Découpage en 25 parties					
Fusion-fission	5,28	40 725	1,35	41	13
Scotch	5,67	42 596	1,38	39	13
Graclus	6,26	48 040	1,46	36	12
Paramètres	Scotch : $bal = 0,11$ Fusion-fission : $coût\_min = 20$, $nombre\_de\_passes = 56$, $\alpha = 0,00024$				
Découpage en 31 parties					
Centres, le 17/6/2005	8,82	48 465	3,29	50	2
Fusion-fission	7,43	47 574	1,38	34	1
Scotch	7,69	45 700	1,58	42	7
Graclus	8,24	51 797	1,48	31	6
FF 29 parties	7,08	46 900	1,29	34	13
Paramètres	Scotch : $bal = 0,06$ Fusion-fission : $coût\_min = 22$, $nombre\_de\_passes = 56$, $\alpha = 0,00026$				
Découpage en 50 parties					
Zones, le 17/6/2005	20,74	75 371	3,11	42	2
Fusion-fission	14,40	56 824	1,73	23	1
Scotch	15,49	58 678	1,74	31	4
Graclus	17,18	65 245	1,71	25	6
Paramètres	Scotch : $bal = 0,07$ Fusion-fission : $coût\_min = 26$, $nombre\_de\_passes = 56$, $\alpha = 0,00026$				

TAB. 5.7 – Résultats du découpage de la *core area* européenne en 25, 31 et 50 parties.

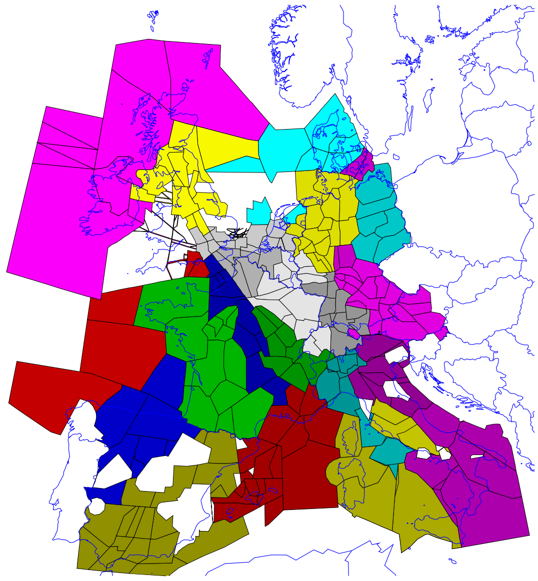
Le tableau 5.7 synthétise tous les résultats du découpage de la *core area* en 25, 31 et 51 parties, ainsi que les données du découpage actuel. Ce tableau se décompose en trois parties :

– la première partie concerne le découpage de la *core area* en 25 parties. La partition trouvée par le logiciel Graclus possède des caractéristiques beaucoup moins intéressantes que celles trouvées par la fusion-fission et par Scotch. Pour une balance de partitionnement légèrement inférieure, la partition trouvée par la fusion-fission a un ratio et un coût de coupe plus bas que ceux de la partition trouvée par le logiciel Scotch. Le nombre de secteurs maximum de cette dernière est légèrement inférieur à celui de la première, mais les deux sont largement au-dessous des recommandations d'Eurocontrol. Quant à leurs nombres de secteurs minimum, ils sont égaux ;
– la première ligne de la seconde partie de ce tableau évalue les caractéristiques du découpage actuel de la *core area* en centres de contrôles. Suivent les partitions en 31 parties trouvées par la fusion-fission, et par les logiciels Scotch et Graclus. Le ratio de coupe de la partition trouvée par la fusion-fission est inférieur aux autres partitions, et sa balance est beaucoup moins élevée. Cependant, pour la première fois (c'est la seule), un coût de coupe inférieur à celui de la fusion-fission a été trouvé. C'est le logiciel Scotch, qui a pour but de minimiser le coût de coupe, qui a trouvé celui-ci. Cependant cela c'est fait au détriment du nombre maximal de secteurs dans une partie, qui vaut 42 dans le cas de Scotch, mais 23 pour la fusion-fission et 25 pour Graclus. Enfin, le nombre du minimum de secteurs, 1, de la partition trouvée par la fusion-fission s'explique par les caractéristiques particulières de ce secteur qui a un coût de coupe très petit (peu d'avions de ce secteur vont dans un autre secteur) et un poids très grand (beaucoup d'avions passent dans ce secteur en provenance et/ou à destination de zones d'espace hors de notre champ d'étude). Ainsi, ce secteur a un ratio de coupe si faible que l'intégrer dans une autre partie ferait beaucoup augmenter le ratio de coupe de la partition. Cependant, une partie, *i.e.* une zone de qualification, d'un seul secteur n'a aucun sens réel, cela montre les limites de l'utilisation du ratio de coupe. La seconde partie la plus petite contient elle 11 secteurs, soit bien plus que dans le cas des partitions de Scotch et Graclus. Grâce à la particularité qu'a la fusion-fission de trouver des partitions de cardinaux différents en une seule exécution, nous pouvons vérifier si les partitions de cardinaux inférieurs à 31 possèdent aussi une partie d'un seul élément. C'est le cas pour la partition en 30 parties, mais pas pour celle en 29 parties. Comme cette dernière possède des caractéristiques intéressantes, elle est incluse après les résultats de Graclus. Il faut souligner que le découpage actuel en centres a un coût de coupe proche de ceux trouvés par la fusion-fission ou Scotch, et même meilleur que celui trouvé par Graclus. Cependant, cela est dû à une balance de partitionnement extrêmement élevée ;
– la première ligne de la dernière partie du tableau présente les caractéristiques du découpage de la *core area* en zones de qualifications, puis suivent les découpages trouvés par la fusion-fission, Scotch et Graclus en 50 parties. Pour des balances de partitionnement très proches, c'est la fusion-fission qui trouve la partition de ratio et de coût de coupe les plus petits. C'est aussi cette partition qui a le nombre maximum de secteurs dans une partie le plus faible. Cependant, nous retrouvons là encore une partie ne possédant qu'un seul secteur. Il s'agit du même secteur que celui de la partition en 31 parties. Dans ce cas, il faut attendre une partition en 45 parties pour que ce secteur ne soit plus isolé.

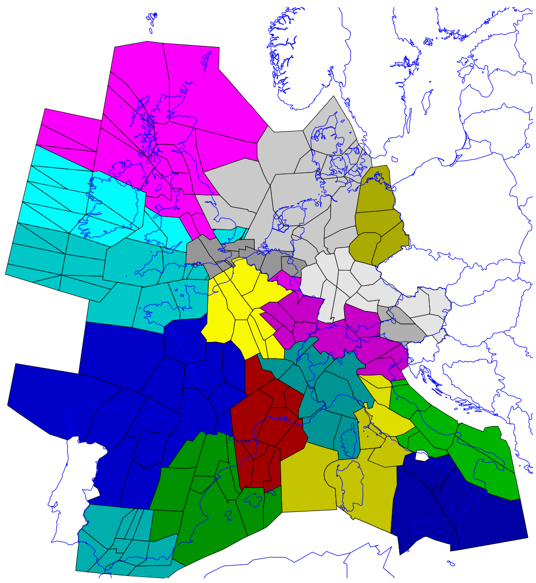
La figure 5.13 présente les découpages en 25 parties trouvés par la fusion-fission (à gauche, images 5.13(a) et 5.13(c)) et l'outil Scotch (à droite, images 5.13(b) et 5.13(d)). Quelques commentaires s'imposent concernant les espaces blancs présents sur ces figures au niveau de vol FL200. Ceux entourant l'Angleterre, sur la mer baltique et la Manche sont dus au fait que ces régions sont des zones d'espace d'information et non de contrôle au sens donné par l'OACI (*supra* 5.1.2). Ceux au-dessus de



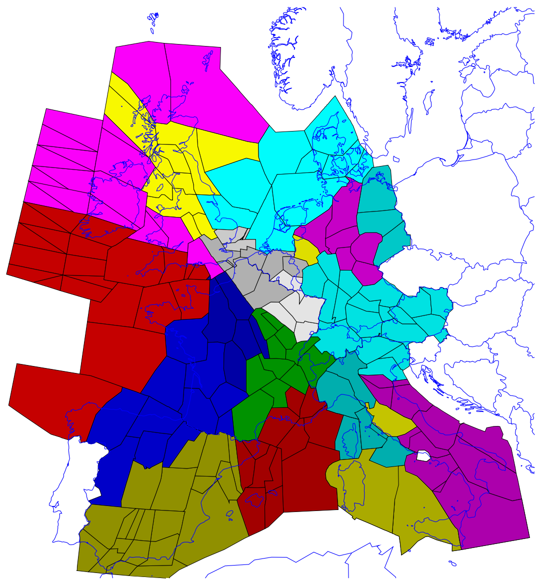
(a) Découpage en 25 parties trouvé par la fusion-fission au FL200.



(b) Découpage en 25 parties trouvé par Scotch au FL200.

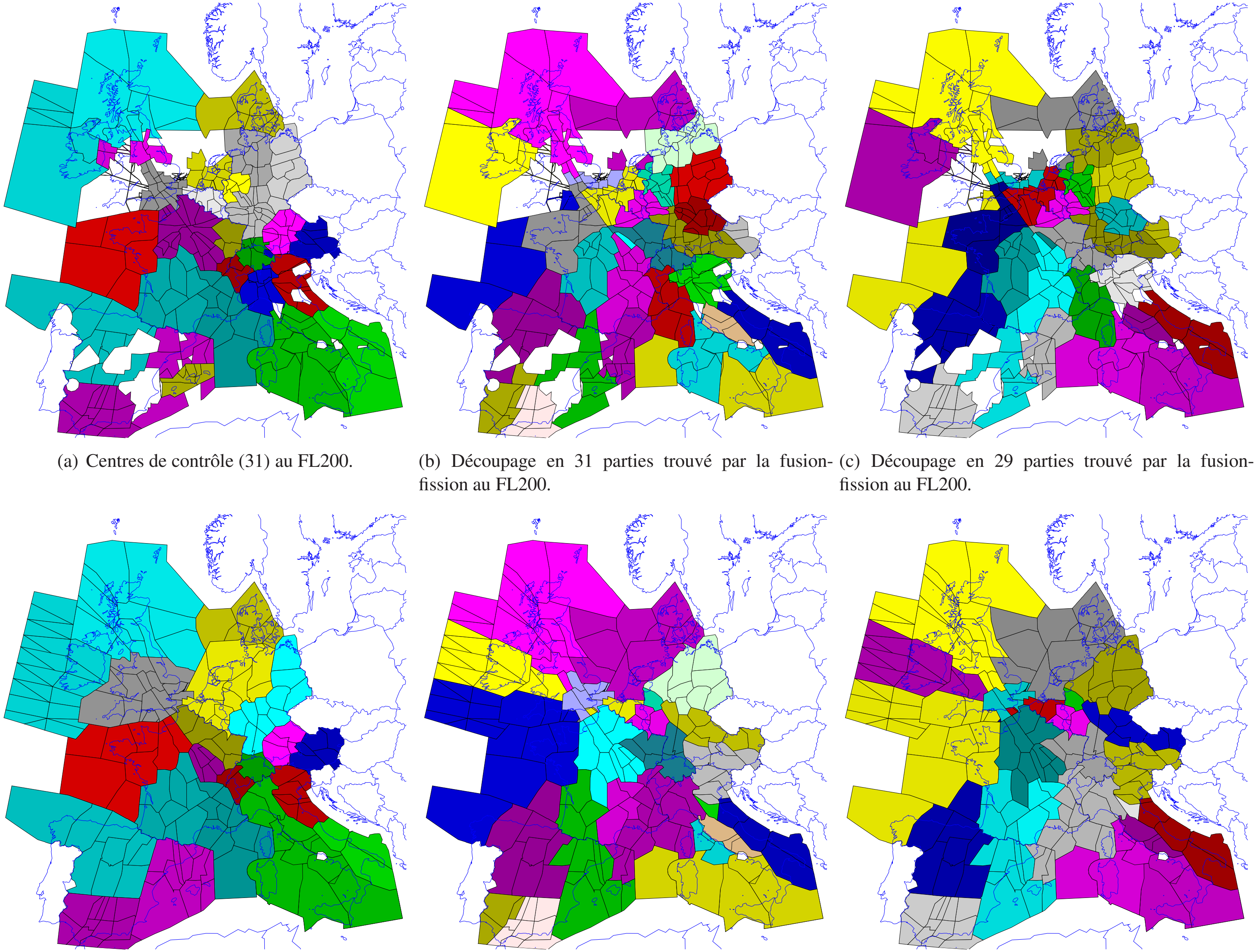


(c) Découpage en 25 parties trouvé par la fusion-fission au FL400.



(d) Découpage en 25 parties trouvé par Scotch au FL400.

FIG. 5.13 – Résultat pour le découpage en 25 parties de l'espace aérien de la *core area*.



(a) Centres de contrôle (31) au FL200.

(b) Découpage en 31 parties trouvé par la fusion-fission au FL200.

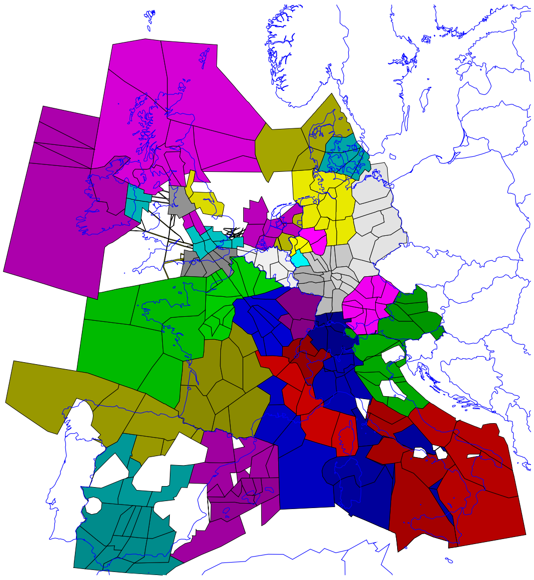
(c) Découpage en 29 parties trouvé par la fusion-fission au FL200.

(d) Centres de contrôle (31) au FL400

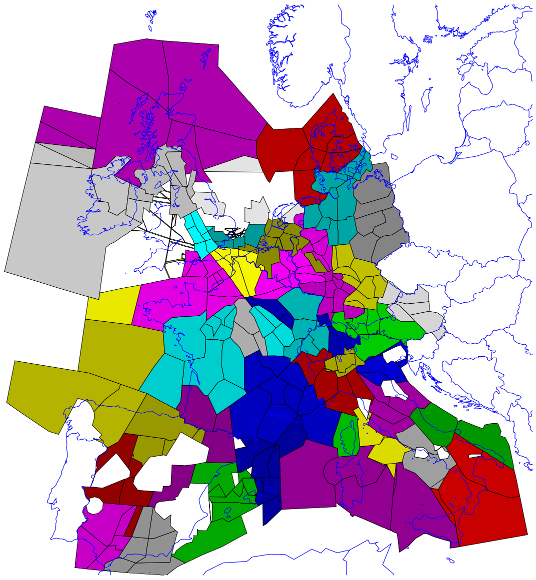
(e) Découpage en 31 parties trouvé par la fusion-fission au FL400.

(f) Découpage en 29 parties trouvé par la fusion-fission au FL400.

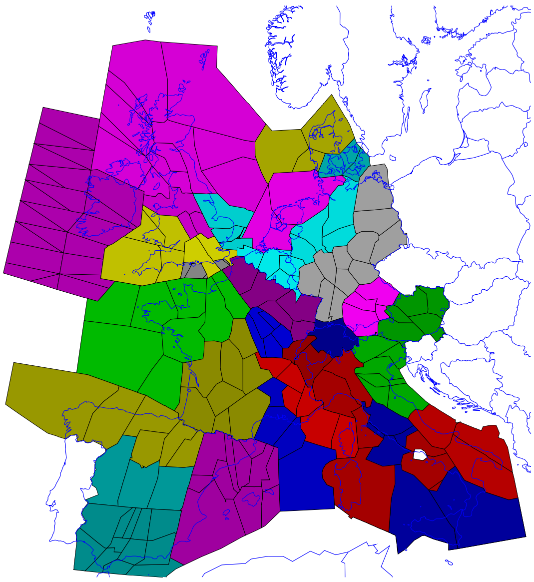
FIG. 5.14 – Représentation des 31 centres de contrôle de la *core area* et du découpage en 31 et 29 parties de celle-ci trouvé par la fusion-fission.



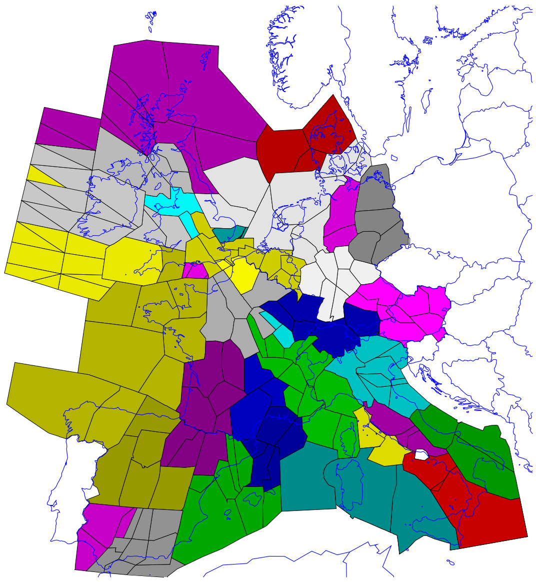
(a) Zones de qualification (50) au FL200.



(b) Découpage en 50 parties trouvé par la fusion-fission au FL200.



(c) Zones de qualification au FL400.



(d) Découpage en 50 parties trouvé par la fusion-fission au FL400.

FIG. 5.15 – Représentation des 50 zones de qualification de la *core area* et du découpage en 50 parties de celle-ci trouvé par la fusion-fission.

l'Espagne et de l'Italie correspondent aux emplacements des secteurs de contrôle des approches des grands aéroports espagnols et italiens, et ne font donc pas partie de centres de contrôle.

Les découpages trouvés par la fusion-fission et Scotch ont plusieurs points communs. Les parties en Allemagne, au Danemark, du sud-ouest de la France ou du sud de l'Espagne se ressemblent. Les parties dans l'ouest de la figure sont dans les deux cas orientés vers l'océan Atlantique et donc vers les flux venant de l'Amérique.

La figure 5.13 présente trois découpages : le découpage réel en 31 centres de contrôles (images 5.14(a) et 5.14(d)), et les partitions trouvées par la fusion-fission en 31 (images 5.14(b) et 5.14(e)) et 29 parties (images 5.14(c) et 5.14(f)). Ces deux dernieres se ressemblent beaucoup. La différence principale qui peut être faite est que celle en 29 parties contient des parties fusionnées par rapport à celle en 31 parties. Les partitions trouvées par la fusion-fission présentent à l'ouest des parties orientés par les flux venant de l'Amérique. Au sud, nous pouvons observer une partie orientée par les flux entre la France et l'Afrique du Nord, alors qu'il est aussi le prolongement d'une partie allant vers l'Italie. Le nord du Royaume-Uni forme une grande partie isolée qui ressemble au centres de contrôle existant, et un peu plus à l'est, le Danemark et les Pays-Bas sont dans une partie commune aux niveaux supérieurs (au-dessus du FL285). On remarque encore que Paris et Francfort sont, aux niveaux inférieurs (avant le FL285), aux carrefours de nombreuses parties. Le découpage réel est quand à lui assez différent de celui trouvé par la fusion-fission.

La figure 5.15 rassemble le découpage réel en 50 zones de qualification (images 5.15(b) et 5.15(d)) et celui trouvé par la fusion-fission (images 5.15(b) et 5.15(d)) en 50 parties. Ce dernier découpage semble un peu désordonné. En effet, minimiser le ratio de coupe pour un nombre de parties aussi important ne fait plus apparaître des parties aussi ordonnées que précédemment, dans le sens vertical comme horizontal, car les flux locaux deviennent plus importants que les flux globaux (ce commentaire est aussi valable pour le coût de coupe). Cependant, un certain nombre de commentaires fait pour le découpage en 31 parties reste valable pour celui-ci. À l'ouest, les parties sont orientées vers l'océan Atlantique. Les parties vers le sud sont orientées est-ouest, et celles juste au-dessus, nord-sud. Aux niveaux inférieurs, Paris et Francfort sont les points de convergence de plusieurs parties.

Les découpages trouvés pour un grand nombre de parties nous montrent que, dans ceux-ci, le découpage se fait en fonction des flux locaux et non plus en fonction de flux globaux, d'où l'apparente incohérence de certaines parties trouvées. Au contraire, le découpage en un nombre de parties plus restreint respecte beaucoup plus ces flux globaux.

5.5 Conclusion

Ce chapitre présente les résultats concernant l'application principale de cette thèse, le problème du découpage de l'espace aérien européen en zones de qualification. Après avoir présenté la structure de l'espace aérien avec ses réseaux de route, le rôle du contrôleur et la sectorisation de l'espace, le problème du découpage de l'espace aérien en Europe a été expliqué. Celui-ci s'inscrit dans le projet de création de blocs fonctionnels d'espace en Europe. Puis nous avons détaillé le processus de modélisation de ce problème vers celui du partitionnement de graphe non contraint. Ce processus passe par la compréhension de la notion de charge de contrôle d'un contrôleur, par la description des objectifs et contraintes de notre problème, par l'analyse et le traitement des données de trafic et enfin par la création du graphe du trafic aérien.

Une fois le problème du découpage de l'espace aérien décortiqué et analysé, une série de résultats a été proposé. Les premiers résultats présentés ont eu pour but de sélectionner les outils de partitionnement les plus adaptés pour le découpage de l'espace aérien. Ainsi, ont été retenus, parmi les métaheuristiques, la méthode de fusion-fission, et parmi les outils multi-niveaux, les outils Scotch et Graclus. Puis trois séries de découpages sont proposés. La première série concerne l'espace aérien français en 5 et 7 zones de qualification. La seconde concerne le FAB d'Europe centrale et 12, 16 et 26 zones de qualification. La troisième concerne la *core area* en 25, 31 et 50 zones de qualification. Chacun de ces découpages est proposé en fonction du nombre actuel de zones de qualification, de centres de contrôle, mais aussi des recommandations d'Eurocontrol.

Le but de cette étude n'est pas de proposer un découpage directement opérationnel de l'espace aérien européen en zones de qualification, mais de proposer des outils pour permettre un tel découpage. Dans la perspective d'obtenir des résultats opérationnels, il serait important de réaliser un étude de terrain concernant les difficultés de contrôle dans chaque secteur.

Les découpages proposés dépendent de la forme des secteurs, en particulier, ceux-ci ont été dessinés en fonction des frontières nationales. Un redécoupage plus performant devrait être basé sur un nouveau découpage de l'espace en secteurs de contrôle. De même, le redécoupage proposé est basé sur les *routes aériennes* existantes. Une modification de ces routes aériennes permettrait d'obtenir un redécoupage plus pertinent au niveau opérationnel.

Chapitre 6

Application de la méthode de fusion-fission à d'autres problèmes de partitionnement non contraints

Ce chapitre étudie l'application de la méthode de fusion-fission à deux problèmes de partitionnement non contraints : la segmentation d'image (*infra* 6.1) et la classification de documents (*infra* 6.2).

Cette étude a pour but de comparer les performances de la fusion-fission par rapport aux méthodes de la littérature, sur d'autres problèmes de partitionnement non contraint que celui du découpage de l'espace aérien. Ainsi, afin de permettre une comparaison objective, les modélisations présentées dans la littératures sont utilisées en l'état. Le but de cette étude n'est pas de proposer de nouvelles modélisations, ni de modifier celles utilisées, mais de vérifier la performance de la fusion-fission en comparant tels quels ses résultats avec ceux de la littérature.

6.1 La segmentation d'images

6.1.1 Présentation du problème

Le but de la segmentation d'images est d'isoler des régions homogènes du reste de l'image, en s'assurant qu'elles sont bien différentes les unes des autres. Le but est donc à la fois de découper l'image en des parties homogènes, mais aussi de rassembler les parties les plus homogènes entre elles. Ces deux objectifs sont complémentaires et utiles afin d'obtenir une représentation visuelle satisfaisante de chaque partie de l'image segmentée. L'une des finalités de la segmentation d'images est la reconnaissance de formes, d'objets ou de personnes.

Cependant, la question du choix du critère de segmentation permettant d'obtenir la « meilleure partition » de l'image reste ouverte. De nombreux critères ont été proposés. Nous avons choisi le critère de la coupe normalisée ou *normalized cut* en anglais [SHI00, SHI98b]. Ce critère est largement utilisé dans la littérature [BEN05, MAR04, SHE03, WEI99]. Il est à la fois simple et performant, et il permet d'identifier des zones homogènes qui peuvent être de superficies variables, et peu connectées entre elles.

Pour segmenter une image, un premier traitement consiste à convertir cette image en une matrice de pixels. Différentes caractéristiques de l'image peuvent être utilisée pour créer cette matrice : la luminance (intensité lumineuse), les différences de couleurs, la texture, ou encore une combinaison de ces caractéristiques. Nous n'étudierons les images que par la luminance, ce qui nous permet de ne

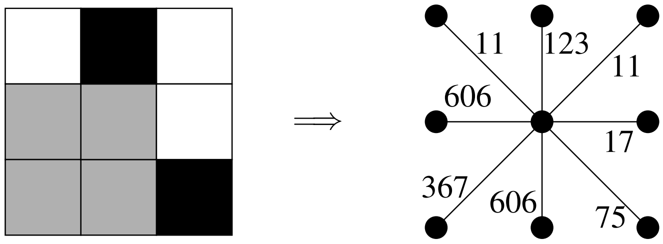


FIG. 6.1 – Conversion d'un pixel en un sommet d'un graphe.

nous intéresser qu'aux images en niveaux de gris.

De manière plus formelle, le graphe $G = (V, E)$ d'une image est construit en considérant chaque pixel comme un sommet du graphe. Une arête du graphe relie deux sommets si la distance séparant les deux pixels respectifs est inférieure à une distance minimale. Chaque arête ainsi constituée entre un sommet i et un sommet j, est pondérée par un poids :

$$poids(i,j) = \exp^{\frac{-\|X(i)-X(j)\|_2^2}{\sigma_X}} * \exp^{\frac{-\|I(i)-I(j)\|_2^2}{\sigma_I}},$$

avec $\| X(i) - X(j) \|_2$ la norme euclidienne de la distance entre les pixels i et j, et $\| I(i) - I(j) \|_2$ la norme pondérée entre 0 et 1 de la différence d'intensité lumineuse entre les pixels i et j.

La figure 6.1 illustre notre conversion d'un pixel en un sommet d'un graphe. Le gris utilisé a une différence normalisée d'intensité de $0,4$ avec le noir et de $0,6$ avec le blanc. Dans cette représentation et pour toutes les figures que nous avons segmentées, nous avons utilisé $\sigma_X = 2$ et $\sigma_I = 0,1$.

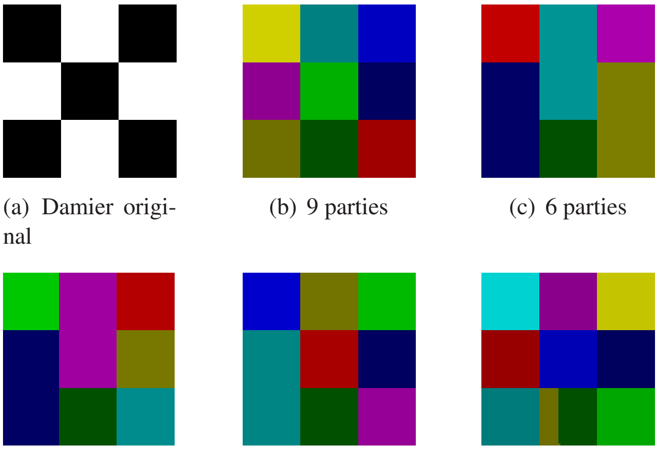
Une étude montre que le nombre de régions différentes isolées dans une image par un humain varie fortement en fonction des personnes à qui l'on demande de faire cette segmentation comme le montre [MAR01]. Ainsi, il n'existe pas *a priori* de nombre k pour lequel il faut segmenter une image particulière.

6.1.2 Application de la méthode de fusion-fission

Le domaine de la segmentation d'images est très actif depuis de nombreuses années, et de nombreuses méthodes ont été développées [FOR03] : la segmentation basée sur les régions, avec les centres mobiles (en anglais *k-means*), la décomposition/agrégation (ou *split and merge*), la croissance de région (ou *region growing*) ; la segmentation basée sur les contours (*edge-based techniques*) ; ou les approches globales comme les histogrammes. Ainsi, des méthodes très générales comme les métaheuristiques peuvent être appliquées à ce problème.

Cette section a pour but d'appliquer la méthode de fusion-fission à la segmentation d'image. Nous avons vu qu'une image pouvait être transformée en graphe, et que ce graphe pouvait alors être partitionné en utilisant la fonction de coût normalisée. Puisqu'une image segmentée possède des régions dont les tailles n'ont aucune raison d'être identiques, le problème de la segmentation d'images peut être résolu en utilisant des méthodes pour le partitionnement non contraint (*supra* 1.6).

Nous avons donc utilisé la méthode de fusion-fission appliquée au partitionnement non contraint pour résoudre le problème de la segmentation d'image. Or, le nombre de partitions créé par la fusion-fission varie au cours du processus d'optimisation. Cette particularité de la méthode de fusion-fission la rend intéressante dans le cas de la segmentation d'images puisque, comme nous l'avons vu, le nombre de parties à créer est en général relatif. Pouvoir choisir après une exécution de l'algorithme entre plusieurs images segmentées peut être appréciable.



(a) Damier original (b) 9 parties (c) 6 parties

(d) 7 parties (e) 8 parties (f) 10 parties

FIG. 6.2 – Damier de 9 cases à partitionner en 9 parties, ainsi que les autres partitions trouvées pendant la même exécution de l'algorithme de fusion-fission.

Le but de cette application à la segmentation d'images n'est pas de chercher à trouver des images « mieux » segmentées que celles proposées dans les articles de la littérature, ni de faire une comparaison poussée avec les techniques existantes, mais plutôt de montrer que l'algorithme de fusion-fission peut être utilisé dans le cas de la segmentation d'image. Nous n'avons pas cherché à optimiser notre algorithme pour la segmentation d'image. De même, le paramétrage de la transformation d'une image en un graphe est assez délicat, cependant, les transformations que nous avons utilisées sont basiques. Une étude plus poussée sur cette adaptation pourrait être réalisée plus tard.

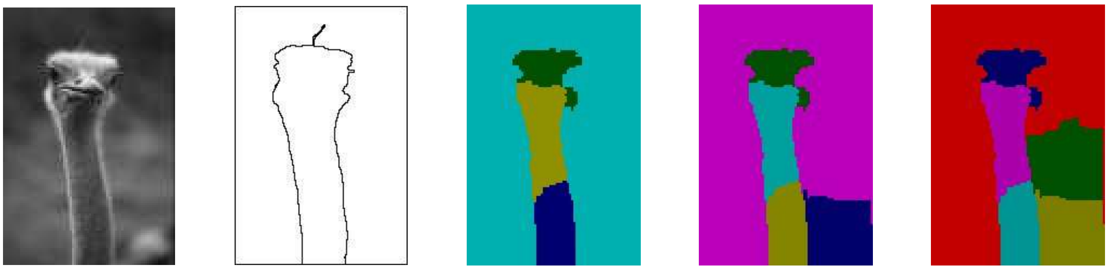
6.1.3 Tests

Notre premier essai de segmentation d'images a été sur un damier. Il avait pour but de savoir si la méthode de fusion-fission pouvait être adaptée à la segmentation d'image, et dans le cas où cela s'avérait possible, de paramétrer l'algorithme. La figure 6.2 représente le découpage d'un damier de 9 cases obtenu après une exécution de notre méthode. Le découpage en 9 parties de ce damier étant parfait, ce premier essai s'est révélé concluant, bien que l'exemple soit très simple. Nous pouvons aussi observer que les découpages en 7 et 8 parties sont eux aussi bons. Le découpage du damier en 10 parties montre, comme on peut s'y attendre, que 8 cases sont bien segmentées et que la 9^{e} est découpée 2 deux parties par un trait droit (dans notre adaptation d'un image en un graphe, une coupe droite possède un coût de coupe plus petit entre deux parties qu'une coupe diagonale).

Après ce premier essai concluant, nous avons cherché à segmenter d'autres images plus complexes. Pour cela, nous avons choisi quatre photographies en niveaux de gris issues d'une base de données d'images de l'université de Berkeley, qui est disponible à l'adresse :
www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/

Cette base de données possède des images en couleur comme en niveaux de gris, et à chaque image sont associées plusieurs segmentations. Celles-ci ont été réalisées grâce à un logiciel de dessin, par des étudiants de Berkeley. En considérant ces segmentations, on remarque que, pour une même image, elles possèdent un nombre de parties différent. Pour illustrer les découpages réalisés par la fusion-fission, nous les comparerons tous avec une de ces segmentations.

La première photographie découpée par l'algorithme de fusion-fission est celle d'une tête d'au-



(a) Photographie originale (b) Découpage fait à la main (c) 4 parties (d) 5 parties (e) 6 parties

FIG. 6.3 – Segmentation de la photographie d'une tête d'autruche par l'algorithme de fusion-fission.

truche (figure 6.3(a)). Une des segmentations proposées dans la base de données de Berkeley est présentée figure 6.3(b). Nous proposons plusieurs découpages trouvés simultanément par l'algorithme de fusion-fission : figures 6.3(c) à 6.3(e). Le fond de cette image est flou et comporte un dégradé progressif, ce qui explique qu'il forme un bloc assez homogène, particulièrement sur la figure 6.3(c). Sur la photographie, le cou de l'autruche semble ne pas avoir de discontinuité jusqu'au bec de l'autruche. Mais l'algorithme de fusion-fission le découpe toujours en deux parties. La tête de l'autruche est séparée de son cou dans les partitions trouvées par notre algorithme, cependant la segmentation de référence (image 6.3(b)) ne le fait pas. Ce qui est le plus surprenant à première vue est que les yeux de l'autruche sont inclus dans le fond de l'image dans les figures 6.3(c) à 6.3(e). Cependant, en regardant l'image d'origine, on s'aperçoit que les yeux de l'autruche sont d'un niveau de gris beaucoup plus proche du fond de l'image que de celui de la tête de l'autruche, ce qui explique le résultat trouvé par notre algorithme. Pour autant, cela souligne l'imperfection de l'approche que nous avons eue qui consiste à créer le graphe en se basant uniquement sur la luminance de ses pixels.

Paramétrage de la fusion-fission pour la segmentation de la figure 6.3(a) :

- $rfusion = rfission = \{0,85; 0,10; 0,04; 0,01\}$;
- $pas\_de\_règle = 0.002$ et $fission\_chaîne = 1,7$;
- $p = 0,5$, $q = 0,8$;
- $coût\_min = 15$ et $nombre\_de\_passes = 32$.

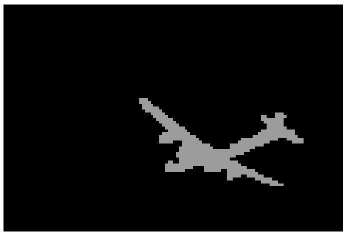
Ce paramétrage a été obtenu empiriquement en suivant les réglages présentés en sous-section 4.3.9. Par la suite, seul ces deux derniers paramètres seront modifiés. Nous pouvons faire remarquer que plusieurs paramètres sont identiques à ceux utilisés pour le découpage de l'espace aérien présenté en section 5.4.1 : $rfusion$, $rfission$, p et q.

La seconde et la troisième photographie segmentées représentent respectivement un avion valant dans un ciel chargé de nuages (figure 6.4(a)) et un bosquet entouré d'un champ (figure 6.5(a)). Ces deux images comportent typiquement deux parties : dans le premier cas, un avion et le fond nuageux comme le découpage manuel le souligne (figure 6.4(b)) ; dans le second cas, un bosquet et les champs l'entourant, bien que le découpage manuel délimite aussi les champs alentours, ce qui nous semble être un découpage beaucoup plus subjectif (figure 6.5(b)). La fusion-fission a, dans les deux cas, assez bien isolé à la fois l'avion (figure 6.4(c)) et le bosquet (figure 6.5(c)). Le découpage de l'avion fait même apparaître les deux moteurs, la dérive et le gouvernail. Quant à la découpe du bosquet, elle fait bien apparaître les débordements de certains arbres sur les champs. Ces deux segmentations sont les plus réussies parmi celles que nous avons obtenues.

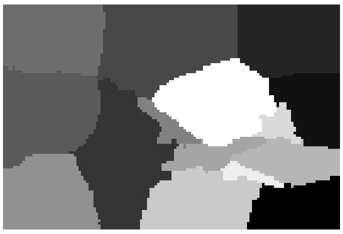
La photographie aérienne a servi d'illustration à d'autres techniques de segmentation. Cette photographie est segmentée en utilisant l'algorithme *generalized belief propagation* (GBP) dans [SHE03].

(a) Photographie originale

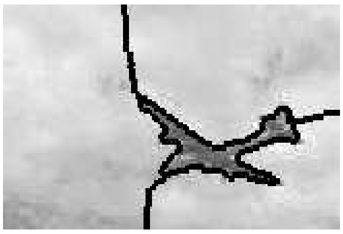
(b) Découpage fait à la main



(c) Fusion-fission en 2 parties

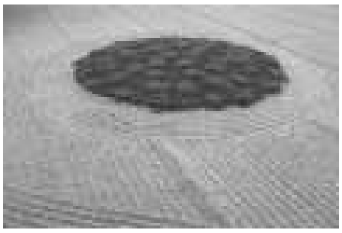


(d) GBP coupe normalisée

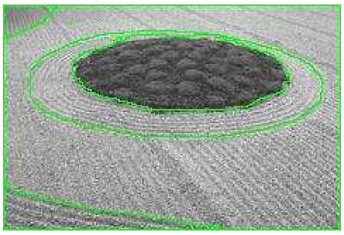


(e) Découpe spectrale

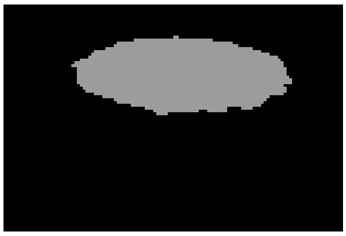
FIG. 6.4 – Segmentation de la photographie d'un avion dans le ciel.



(a) Photographie originale



(b) Découpage fait à la main



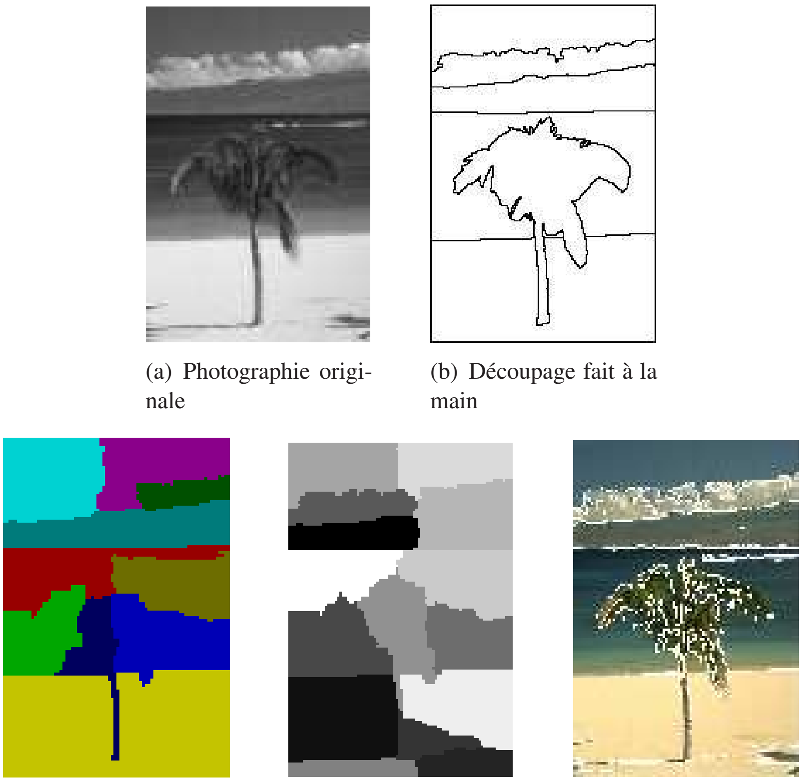
(c) Fusion-fission en 2 parties

FIG. 6.5 – Segmentation de la photographie d'un bosquet entouré de champs.

Cet algorithme permet la minimisation de deux critères au choix : la coupe normalisée ou le « coût typique » (*typical cut*). La figure 6.4(d) représente le résultat trouvé par l'algorithme GBP utilisant la coupe normalisée. Sur cette figure, l'avion est à peine discernable. En comparaison, le découpage trouvé par la fusion-fission (figure 6.4(c)) semble bien meilleur. Nous devons cependant souligner que dans cet article, l'algorithme GBP trouve un découpage beaucoup plus réussi en minimisant le coût typique. Dans [YU03], la photographie de cet avion a été segmentée en utilisant un méthode spectrale. Le résultat de cette segmentation (figure 6.4(e)) s'il est moins précis que celui de la fusion-fission car il comporte un plus grand nombre de parties, permet néanmoins d'isoler l'avion du paysage nuageux.

Nous avons testé aussi une photographie beaucoup plus complexe, celle d'un palmier (figure 6.6(a)). Le découpage réalisé à la main représente clairement le palmier en premier plan avec le sable en bas, surmonté de la mer, puis de la colline, elle-même dépassée par les nuages, puis un ciel clair. Le résultat obtenu par l'algorithme de fusion-fission est beaucoup moins clair (figure 6.6(c)). Le palmier se devine, mais c'est tout, la mer est constitué de 4 blocs qui se mélangent avec le palmier. Cependant la coupure entre la mer et la plage est nette, ainsi qu'entre la mer et la colline. Si une partie des nuages est distinct du ciel clair, le reste est mélangé à celui-ci. Le découpage de cette photographie montre la limite actuelle de notre adaptation.

De même que pour la photographie de l'avion, la photographie de ce palmier a été segmentée



(a) Photographie originale

(b) Découpage fait à la main

(c) Fusion-fission en 10 parties

(d) GBP avec la coupe normalisée

(e) GBP avec le coût typique

FIG. 6.6 – Segmentation de la photographie d'un palmier.

par l'algorithme GBP. La figure 6.6(d) montre le résultat de cet algorithme. Cette fois-ci, les deux segmentations semblent être de qualité équivalentes. Mais, cela veut aussi dire que ni l'une ni l'autre n'est satisfaisante. Si l'on en juge par le résultat obtenu de la segmentation utilisant le coût typique (figure 6.6(e)), alors on peut se demander si on ne devrait pas changer de modèle et adapter la fusion-fission à cette fonction de coût typique qui semble, à première vue, bien meilleure.

Paramétrage de la fusion-fission pour la segmentation des figures 6.4(a) et 6.6(a) : $coût\_min = 24$ et $nombre\_de\_passes = 30$, et pour la figure 6.5(a) : $coût\_min = 15$ et $nombre\_de\_passes = 24$. Le temps d'exécution de l'algorithme de fusion-fission est de 5 minutes à chaque fois.

6.2 La classification de documents

6.2.1 Présentation

Le but de la classification de documents est de regrouper par catégories des textes mélangés dans une même base de données. L'approche classique utilisée pour résoudre ce problèmes se décompose en deux étapes : la première consiste à analyser chaque document pour en extraire son vocabulaire significatif, et la seconde trie les documents à partir des similitudes de vocabulaire utilisées dans ceux-ci.

La méthode de classification utilisée dans cette section est celle décrite par Inderjit Dhillon dans [DHI01]. Dans celui-ci, il présente une méthode permettant de classer en parallèle les documents et leurs mots. Pour cela, une matrice M de correspondance documents-mots est créée. Un élément M_{ij} de cette matrice signifie que le mot de rang i est présent M_{ij} fois dans le document j. D'une manière générale, un très grand nombre de mots est utilisé, même pour un ensemble de documents de petite taille. Cependant, seul un faible pourcentage de ces mots est utilisé par chaque document, ce qui rend la matrice M très creuse. Dans [DHI01], le pourcentage de valeurs nulles de cette matrice est évalué à 99%.

On distingue la classification supervisée (en anglais *classification*), dont les catégories sont connues *a priori*, de la classification non supervisée (en anglais *clustering*), dont les catégories sont fondées sur la structure propre de la « proximité » des objets. Nous ne nous intéresserons dans cette section qu'à la classification supervisée de documents.

De nombreuses méthodes de classification existent pour résoudre ce problème [JAI99] : la classification hiérarchique, les centres mobiles, les méthodes d'agrégation, la méthode des plus proches voisins, *fuzzy clusetring*, etc. La méthode utilisée dans [DHI01] est basée sur le partitionnement de graphe. Plus précisément, le but est de trouver une partition de coût minimale d'un graphe biparti[1].

6.2.2 Modélisation

Un graphe biparti peut se noter $G = (S_d, S_m, A)$ où S_d est le premier ensemble de sommets, l'ensemble des documents dans notre cas, S_m le second ensemble de sommets qui sera pour nous l'ensemble des mots, et A l'ensemble des arêtes reliant les sommets de S_d à ceux de S_m. Soient deux sommets $s \in S_d$ et $s' \in S_m$, si le nombre s' apparaît dans le document s, alors l'arête (s, s') est incluse dans A et $poids(s, s')$ représente le nombre d'occurrences de s' dans s.

Trouver une partition d'un graphe biparti revient à regrouper dans les mêmes parties des sommets des deux ensembles S_d et S_m. Le but de cette classification de documents va être de partitionner le graphe biparti réalisé après la lecture de l'ensemble des documents présentés, en cherchant à minimiser les liens entre différentes parties. Les parties trouvées seront constituées des associations mots-documents les plus forts, ce qui permettra donc de classer les documents comme les mots. Cependant, la taille des parties recherchées n'est pas connue à l'avance. Ce problème n'est donc pas celui du partitionnement de graphe contraint, mais celui du cas non contraint (*supra* 1.6).

La question soulevée par cette adaptation du problème du partitionnement de graphe est celle de la fonction objectif à utiliser. En effet, comment savoir si la fonction objectif utilisée va permettre de trouver une classification « parfaite » des documents. S'il est difficile de créer une telle fonction objectif, il est possible d'invalider une fonction proposée. Pour cela, il suffit de se procurer plusieurs bases de données de documents de sujets très différents, de les mélanger entre elles, puis de les trier en utilisant la fonction objectif proposée, puis de vérifier si une valeur plus petite de celle-ci par rapport à la classification d'origine ne peut être trouvée.

Dans [JAI99], Inderjit Dhillon propose de se servir de la fonction de coupe normalisée (*supra* 1.4) pour résoudre ce problème de classification. Le but de notre expérimentation est de vérifier si cette fonction objectif est valable pour ce problème.

6.2.3 Extraction des mots d'un document et analyse lexicale

Nous avons récupéré 3 ensembles de documents utilisés dans [JAI99] :

Medline : 1033 résumés médicaux ;

[1] Un graphe biparti est un graphe dont toutes les arêtes relient uniquement deux ensembles distincts de sommets, *i.e.* deux sommets du même ensemble ne peuvent pas être reliés par une arête

Jeux de données	Nb. de documents	Nb. de mots	Nb. d'arêtes du graphe
MedCran	2,433	10,683	129,601
MedCisi	2,493	11,616	118,304
MedCranCisi	3,893	13,192	192,857

TAB. 6.1 – Tailles de graphes créés à partir de différents jeux de données.

Algorithme	partie	Medline	Cranfield	coupe normalisée	balance
Classification d'origine	S_0	1033	0	0,3037	1,22
	S_1	0	1400		
Fusion-fission	S_0	1019	0	0,3030	1,30
	S_1	14	1400		
Graclus	S_0	765	0	0,3706	1,43
	S_1	268	1400		

TAB. 6.2 – Résultats de la bissection du problème MedCran.

Cranfield : 1400 résumés de documents aéronautiques ;

Cisi : 1460 résumés de recherche documentaire.

Cette base de données est disponible à l'adresse suivante : ftp ://ftp.cs.cornell.edu/pub/smart .

Afin de pouvoir créer le graphe biparti documents-mots, l'extraction des mots de ces documents doit être réalisée. Celle-ci se déroule en trois étapes :

1. les mots sont extraits des différents documents à partir de l'analyse des différents séparateurs lexicaux comme les espaces blancs ou les signes de ponctuation ;
2. les « mots vides », ou *stop words* en anglais, sont supprimés de la liste de mots précédemment extraits. Les mots vides sont des mots de liaison (aussi appelés connecteurs logiques), ou des mots tellement courants que leur utilisation n'apporte pas de signification supplémentaire au texte. Nous avons téléchargé et utilisé la liste de mots vides disponibles à la même adresse que celle des bases de données ;
3. enfin, nous avons extrait le lexème[2] de chaque mot de la liste de « mots pleins » restant. Cette étape a été réalisée grâce à l'utilisation d'un logiciel prévu à cet effet et basé sur le *Porter stemming algorithm* [POR80]. Ce logiciel est disponible à l'adresse :
http ://tartarus.org/ martin/PorterStemmer/ .

À partir de la liste de mots (ou plus exactement de lexèmes) trouvée, nous avons pu construire les graphes qui serviront aux classifications présentées dans la section suivante. Les caractéristiques de ces graphes sont décrites dans le tableau 6.1.

6.2.4 Résultats

Nous avons créé quatre jeux de données à partir des trois ensembles de documents de la sous-section précédente. Les trois premiers sont les trois paires possibles de la combinaison de ces ensembles de documents, nous les noterons : MedCran, MedCisi et CranCisi. Le dernier, nommé MedCranCisi, est constitué du mélange des trois ensembles de documents.

[2]Un lexème est une unité minimale de signification appartenant au lexique.

Algorithme	partie	Medline	Cisi	coupe normalisée	balance
Classification d'origine	S_0	1033	0	0,3402	1,04
	S_1	0	1460		
Fusion-fission	S_0	988	4	0,3346	1,15
	S_1	45	1456		
Graclus	S_0	825	40	0,3474	1,21
	S_1	208	1420		

TAB. 6.3 – Résultats de la bissection du problème MedCisi.

Algorithme	partie	Medline	Cranfield	Cisi	coupe normalisée	balance
Classification d'origine	S_0	1033	0	0	0,7586	1,31
	S_1	0	1400	0		
	S_2	0	0	1460		
Fusion-fission	S_0	903	0	4	0,7499	1,22
	S_1	7	1385	8		
	S_2	123	13	1448		
Graclus	S_0	874	0	0	0,7602	1,27
	S_1	18	1384	9		
	S_2	141	14	1451		

TAB. 6.4 – Résultats de la 3-partition du problème MedCranCisi.

Les 3 ensembles de documents, Medline, Cranfield et Cisi, sont approximativement de même taille. Ainsi, la classification d'origine des jeux de données, qui consiste à créer une partition de ceux-ci dans laquelle chaque partie est égale à un ensemble de documents, possède une balance de partitionnement assez petite. Les tableaux 6.2, 6.3 et 6.4 présentent respectivement la classification d'origine des jeux de données MedCran, MedCisi et MedCranCisi. Comme les balances de partitionnement d'origine ne dépassent pas $1,31$, et parce que les graphes engendrés par ces différents jeux de données sont de grandes tailles (*cf* tableau 6.1), nous avons choisi d'utiliser l'algorithme de fusion-fission appliqué au partitionnement de graphe contraint (*supra* 4.4).

Nous avons appliqué sur chaque jeu de données l'algorithme de fusion-fission ainsi que l'outil de partitionnement Graclus (*supra* A). Ce dernier a été choisi car il permet de minimiser la coupe normalisée. En plus, nous avons comparé les résultats trouvés par l'outil Graclus avec ceux du logiciel pMetis. Le tableau 6.5 présente ces résultats, et permet de vérifier que ceux de Graclus sont effectivement plus performants, et que ce logiciel est utile pour résoudre ce problème.

Le tableau 6.2 compare les partitions du jeu de test MedCran, trouvées par les différentes méthodes de partitionnement. Nous pouvons constater que si l'outil Graclus n'a pu trouver une partition de coupe normalisée proche de celle de la classification d'origine, l'algorithme de fusion-fission a trouvé une partition de coupe normalisée inférieure. De plus, c'est au bout de seulement 7 secondes de calcul que la première partition de coupe normalisée plus petite que celle du classement d'origine, a été trouvée.

Le paramétrage de l'algorithme de fusion-fission a été le même pour tous les résultats présentés dans cette section : une balance de partitionnement maximale imposée de $2,00$, et un nombre d'itérations $n = 2000$.

Les tableaux 6.3 et 6.4 présentent les résultats obtenus pour les jeux de tests MedCisi et MedCranCisi. Dans les deux cas, l'outil Graclus trouve des partitions dont la coupe normalisée, si elle est

	MedCran	MedCisi	MedCranCisi
pMetis	0.4729	0.3863	1.1635
Graclus	0.3706	0.3474	0.7602

TAB. 6.5 – Comparaison des coupes normalisées trouvées par les outils pMetis et Graclus.

proche de celle de la classification d'origine, est cependant plus grande que celle-ci. Mais ces deux tableaux montrent aussi que l'algorithme de fusion-fission permet à chaque fois de trouver des partitions dont la coupe normalisée est inférieure à celle de la classification d'origine. Cette première constatation nous indique que l'utilisation de la coupe normalisée comme fonction objectif n'est pas optimale.

Les tableaux 6.2, 6.3 et 6.4 présentent aussi la répartition des documents des partitions trouvées en les comparant avec le classement d'origine. Dans le cas de la fusion-fission, cette répartition est peu différente pour le jeu de données MedCran, mais la différence s'accentue pour MedCisi et devient forte dans le cas de MedCranCisi. Cette seconde constatation, ajoutée à la première, nous permet de dire que l'utilisation de la coupe normalisée pour la classification de documents n'est pas optimale.

6.3 Conclusion

Ce chapitre présente deux applications de la méthode de fusion-fission à des problèmes d'optimisation combinatoire modélisables sous forme de problèmes de partitionnement de graphes non contraints. Le premier problème étudié est celui de la segmentation d'image. L'image est modélisée sous la forme d'un graphe, puis partitionnée en fonction du coût de coupe normalisé. La seconde application est celle de la classification de documents. Un graphe est créé en fonction de l'adéquation du vocabulaire présent dans les différents documents.

Ces deux applications ont été réalisées afin d'étudier les performances de la méthode de fusion-fission sur d'autres problèmes que ceux du découpage de l'espace aérien. Afin que cette étude soit aussi objective que possible, nous sommes restés fidèles aux modélisations sous formes de partitionnement de graphes de ces deux problèmes, tels qu'ils sont proposés dans la littérature. Nous n'avons donc pas cherché à améliorer ces modélisations.

Dans le domaine de la segmentation d'image, il existe d'autres modélisations de ce problème dont les résultats sont plus intéressants que ceux de la modélisation que nous avons utilisée, mais qui ne font pas appel au partitionnement de graphe. Cependant, notre étude montre qu'avec la modélisation utilisée, la fusion-fission permet de trouver des résultats au moins aussi bons que ceux d'autres méthodes de la littérature.

D'autre part, notre étude à aussi permis de montrer, dans le cas de la classification de documents, les faiblesses de la modélisation du problème sous forme de partitionnement de graphe telle qu'elle est présentée dans la littérature. Ainsi, l'utilisation de la coupe normalisée pour modéliser la classification de document n'est pas optimale. Il reste cependant à trouver une fonction objectif plus performante.

Chapitre 7

Application de la méthode de fusion-fission au partitionnement contraint

L'application de la méthode de fusion-fission au problème du découpage de l'espace aérien européen a donné de bon résultats comparativement à d'autres méthodes de partitionnement (*supra* 5). Cependant, ce problème de trafic aérien, qui est un problème de partitionnement non contraint, est assez différent de celui du partitionnement contraint, que cherchent à résoudre les outils et méthodes de partitionnement classiques. La comparaison entre les résultats trouvés par la fusion-fission et ceux trouvés par ces méthodes est donc biaisée pour deux raisons : les contraintes ne sont pas du même ordre et les fonctions objectifs sont différentes.

Fort de ce constat, nous avons voulu adapter la méthode de fusion-fission au problème pour lequel ces outils et méthodes de partitionnement ont été conçues, et non plus l'inverse. Le but de cette adaptation de la fusion-fission au partitionnement de graphe contraint est de pouvoir comparer plus objectivement les résultats des différentes méthodes sur le problème contraint. Nous avons aussi utilisé la fusion-fission sur les bancs de tests classiques de partitionnement de graphe.

Ce chapitre présente les résultats obtenus par la méthode de fusion-fission en les comparant à ceux des outils de l'état de l'art (*supra* A.1). Les graphes qui ont servi au partitionnement sont ceux présents dans les archives du partitionnement de graphe (*infra* annexe A.4). Le tableau 7 récapitule leurs caractéristiques.

Dans une première section, la robustesse de l'algorithme de fusion-fission pour le partitionnement contraint va être comparée avec celle des outils multi-niveaux (*supra* 7.1). Puis, sont étudiés les gains de performances obtenus par la fusion-fission par rapport à une itération des méthodes multi-niveaux (*supra* 7.2). Dans une troisième section, sont comparés les résultats obtenus avec les différentes améliorations apportées à l'algorithme de fusion-fission (*supra* 7.3). La quatrième section compare les meilleurs partitions trouvées actuellement pour les bancs de tests avec ceux obtenus par la fusion-fission (*supra* 7.4). Enfin, la dernière section conclue ce chapitre (*supra* 7.5).

7.1 Robustesse de la fusion-fission

Cette section présente les premiers résultats des bancs de tests trouvés par l'algorithme de fusion-fission. Celui-ci utilise une séquence aléatoire de n entiers suivant une loi binomiale (*supra* 4.4). Comme nous l'avons fait remarquer, cette première version ne permet pas de stopper l'algorithme

	Taille		Degré		
Nom du graphe	$card(S)$	$card(A)$	min.	max.	moyenne
add20	2 395	7 462	1	123	6,23
data	2 851	15 093	3	17	10,59
3elt	4 720	13 722	3	9	5,81
uk	4 824	6 837	1	3	2,83
add32	4 960	9 462	1	31	3,82
bcsstk33	8 738	291 583	19	140	66,74
whitaker3	9 800	28 989	3	8	5,92
crack	10 240	30 380	3	9	5,93
wing_nodal	10 937	75 488	5	28	13,80
fe_4elt2	11 143	32 818	3	12	5,89
vibrobox	12 328	165 250	8	120	26,81
bcsstk29	13 992	302 748	4	70	43,27
4elt	15 606	45 878	3	10	5,88
fe_sphere	16 386	49 152	4	6	6,00
cti	16 840	48 232	3	6	5,73
memplus	17 758	54 196	1	573	6,10
cs4	22 499	43 858	2	4	3,90
bcsstk30	28 924	1 007 284	3	218	69,65
bcsstk31	35 588	572 914	1	188	32,20
bcsstk32	44 609	985 046	1	215	44,16
t60k	60 005	89 440	2	3	2,98
wing	62 032	121 544	2	4	3,92
brack2	62 631	366 559	3	32	11,71
finan512	74 752	261 120	2	54	6,99
fe_tooth	78 136	452 591	3	39	11,58
fe_rotor	99 617	662 431	5	125	13,30
598a	110 971	741 934	5	26	13,37
fe_ocean	143 437	409 593	1	6	5,71
144	144 649	1 074 393	4	26	14,86
wave	156 317	1 059 331	3	44	13,55
m14b	214 765	1 679 018	4	40	15,64

TAB. 7.1 – Caractéristiques des graphes du banc de tests utilisé.

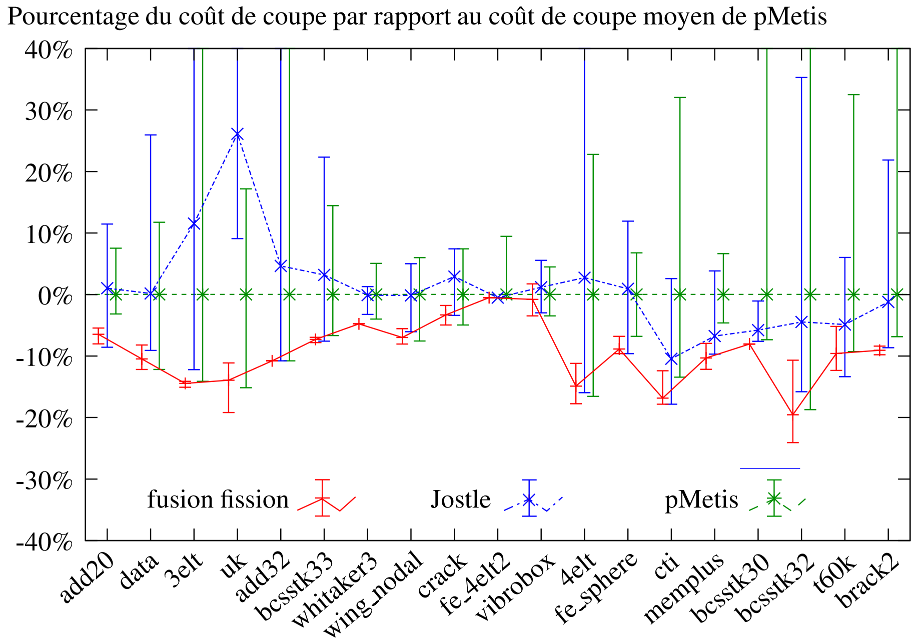


FIG. 7.1 – Coûts de coupe minimum, moyen et maximum des 20 bissections obtenues pour une balance de partitionnement de $1,01$, par la fusion-fission et les logiciels Jostle et pMetis

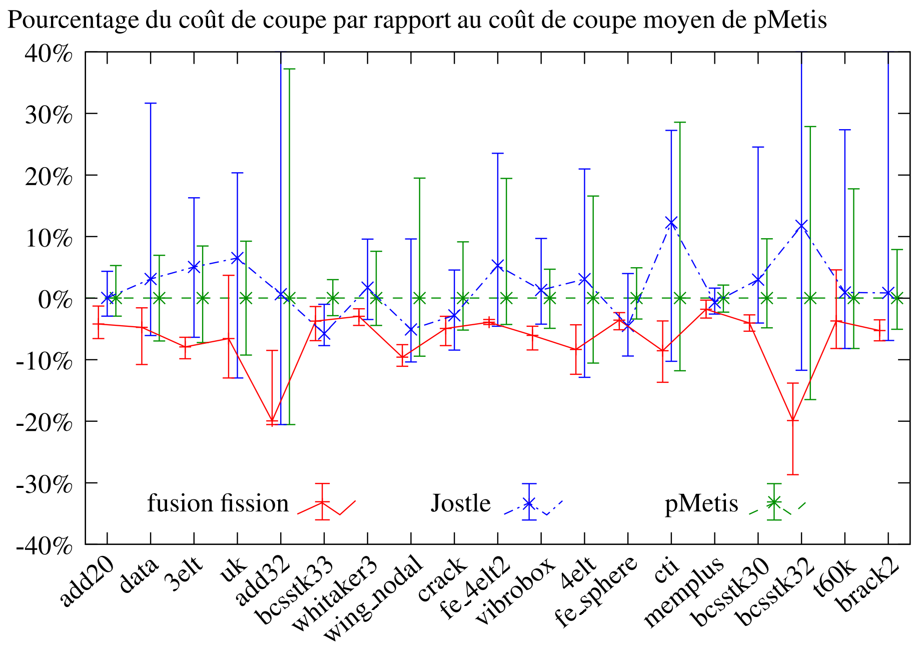


FIG. 7.2 – Coûts de coupe minimum, moyen et maximum des 20 partitions en 4 parties obtenues pour une balance de partitionnement de $1,01$, par la fusion-fission et les logiciels Jostle et pMetis

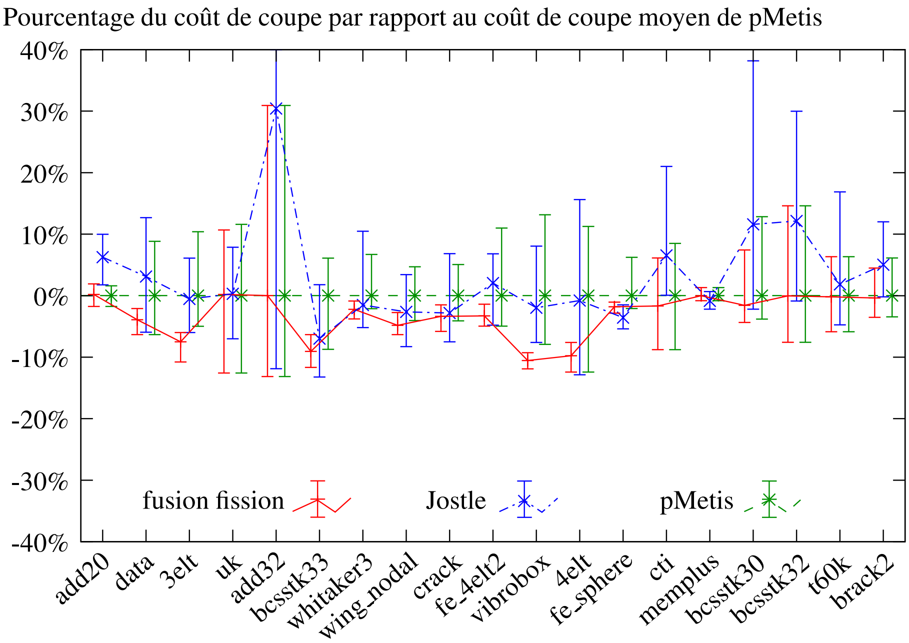


FIG. 7.3 – Coûts de coupe minimum, moyen et maximum des 20 partitions en 8 parties obtenues pour une balance de partitionnement de $1,01$, par la fusion-fission et les logiciels Jostle et pMetis

après un temps d'exécution fixe. Afin ne pas avoir un temps de calcul prohibitif, nous avons limité le nombre d'itérations de la fusion-fission à $n = 1000$.

L'algorithme d'affinage utilisé est le *Global Kernighan-Lin refinement* (GKLR, *supra* 2.5.3). L'algorithme de fusion-fission procède à 4 affinages à chaque itération de sa boucle interne (ou boucle sur le nombre de parties), un pour chaque valeur de la balance de partitionnement des bancs de tests : $1,0$, $1,01$, $1,03$ et $1,05$. Cette adaptation de l'algorithme est spécialement conçue pour ceux-ci, car elle permet de ne pas exécuter l'algorithme de fusion-fission pour chacune de ces balances de partitionnement. Cependant, comme le processus d'affinage est long, le répéter 4 fois de suite allonge significativement la durée d'exécution de l'algorithme.

Les calculs présentés dans cette section et celles qui vont suivre ont été réalisés sur un ordinateur fonctionnant avec le système d'exploitation GNU/Linux Debian, muni d'un processeur Intel Pentium IV cadencé à 3GHz, disposant de 512Mo de mémoire vive.

Les figures 7.1 à 7.3 comparent les coûts de coupe trouvés par 20 exécutions de l'algorithme de fusion-fission et des logiciels Jostle et pMetis. Afin de simuler le changement de clef de la fonction de génération des nombres aléatoires des outils de partitionnement (les sources de Jostle ne sont pas disponibles), les indices des sommets du graphe à partitionner on été permutés, sans que cela change le graphe. Les coûts de coupe présentés sur ces figures sont les coûts de coupe minimum, moyen et maximum de 20 partitions trouvées par chaque outil pour chaque graphe. Comme les coûts de coupe des partitions trouvées sont très différents selon les graphes, les coûts de coupe des partitions de chaque graphe ont été divisés par le coût de coupe moyen de pMetis. Ainsi, un résultat est d'autant plus bon qu'il se situe en dessous de la ligne 0.

L'algorithme de fusion-fission est auto-suffisant dans le sens où il ne nécessite pas d'autre paramètre que le graphe, le nombre de parties de la partition cherchée et le nombre d'itérations de l'algorithme.

Pour des partitions en 2 et 4 parties (figures 7.1 et 7.2), les résultats trouvés par la fusion-

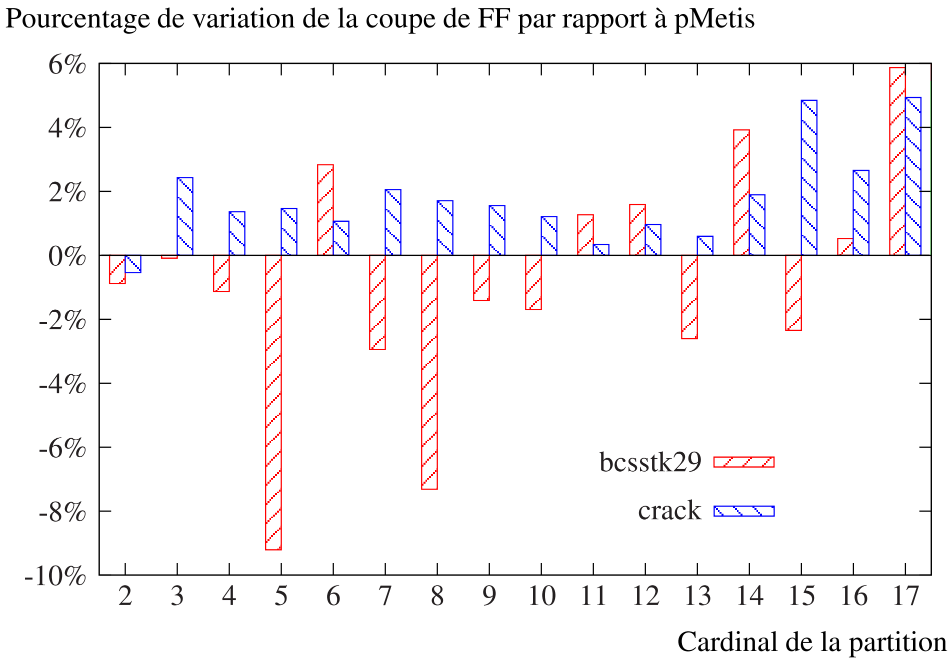


FIG. 7.4 – Comparaison entre les partitions trouvées après une exécution de la fusion-fission pour $k = 8$ et celles trouvées par 16 exécutions (une par nombre de parties) de 30 minutes d'itérations de pMetis. Les résultats sont présentés en fonction de la différence de coût de coupe entre la fusion-fission et pMetis, en pourcentage du coût de coupe de pMetis. Les résultats en dessous de 0% montrent que la fusion-fission a trouvé une meilleure partition que pMetis.

fission sont particulièrement bons. Non seulement les coûts de coupe moyens de la fusion-fission sont inférieurs à ceux des deux logiciels, mais les écarts entre les coûts de coupe maximum et minimum sont aussi beaucoup plus petits que ceux de ces deux logiciels. Cependant, les résultats de la fusion-fission concernant les partitions en 8 parties (figure 7.3) sont plus proches de ceux de Jostle et pMetis. Nous chercherons plus loin à expliquer pourquoi.

Au vu des résultats présentés dans cette section, la méthode de fusion-fission semble beaucoup plus robuste[1] que les logiciels Jostle et pMetis. Ces résultats montrent aussi que la fusion-fission permet de trouver, pour la plupart des graphes, des partitions de coûts de coupe inférieurs à ceux de ces deux logiciels, particulièrement pour des partitions en 2 et 4 parties.

Ces résultats confirment que les méthodes multi-niveaux sont peu robustes. Dans le but d'obtenir de meilleurs résultats, une recherche itérative locale peut être utilisée autour d'une de ces méthodes multi-niveaux. C'est ce que nous proposons dans la section suivante.

7.2 Gains de performance apportés par la fusion-fission

Dans la section précédente, nous avons vu que les méthodes multi-niveaux étaient peu robustes. Bien que cette méthode utilise des boucles itératives dans chacune de ses trois phases, elle est très rapide et peu donc être utilisée au sein d'un algorithme de recherche itérative locale afin d'améliorer

[1]Nous définissons la robustesse d'un algorithme non déterministe, par sa capacité à trouver des résultats de coût proches avec des clefs de génération de nombres aléatoires différents.

	$k=2$		$k=4$		$k=8$	
Graphe	Iter. pMetis	FF_1	Iter. pMetis	FF_1	Iter. pMetis	FF_1
add20	675	**656**	1204	**1197**	**1753**	1800
data	**189**	191	385	**383**	**690**	701
3elt	90	**89**	204	**201**	360	**353**
uk	20	**19**	*44*	*44*	**89**	97
add32	*10*	*10*	*33*	*33*	*66*	*66*
bcsstk33	10169	**10097**	22116	**21662**	35767	**35063**
whitaker3	127	**126**	*383*	*383*	**674**	683
crack	184	**183**	369	**367**	**696**	707
wing_nodal	1709	**1704**	3707	**3628**	5792	**5747**
fe_4elt2	*130*	*130*	*349*	*349*	**624**	630
vibrobox	**10341**	11429	19450	**19418**	25697	**25604**
bcsstk29	2843	**2818**	8514	**8379**	16013	**14926**
4elt	139	**138**	333	**327**	**555**	567
fe_sphere	*386*	*386*	794	**791**	**1248**	1256
cti	334	**318**	953	**948**	**1794**	1860
memplus	5953	**5663**	10073	**9792**	**12357**	12582
Graphe	Iter. kMetis	FF_1	Iter. kMetis	FF_1	Iter. kMetis	FF_1
add20	663	**647**	1194	**1189**	**1755**	1800
data	192	**190**	383	**381**	**664**	690
3elt	*87*	*87*	*199*	*199*	347	**343**
uk	*18*	*18*	**42**	43	**86**	96
add32	*10*	*10*	*33*	*33*	*66*	*66*
bcsstk33	10198	**10064**	22165	**21355**	34968	**34940**
whitaker3	*126*	*126*	389	**381**	**674**	677
crack	184	**182**	375	**366**	707	**704**
wing_nodal	**1688**	1699	3630	**3628**	**5582**	5715
fe_4elt2	*130*	*130*	*349*	*349*	**618**	630
vibrobox	**10368**	11279	**19241**	19398	**24915**	25234
bcsstk29	2851	**2818**	8791	**8237**	16331	**14868**
4elt	138	**137**	**324**	327	**552**	558
fe_sphere	404	**384**	814	**791**	1254	**1253**
cti	331	**318**	998	**934**	1887	**1818**
memplus	5898	**5574**	10059	**9710**	**12411**	12439

TAB. 7.2 – Comparaison entre les coûts de coupe des partitions créées par la fusion-fission, et les meilleurs résultats trouvés après 30 minutes d'itérations des logiciels pMetis, pour une balance de $1,01$, et kMetis, pour une balance de $1,03$

ses performances. C'est ce que nous avons fait dans cette section avec les logiciels pMetis et kMetis. Ceux-ci ont été exécutés pendant 30 minutes en recevant, à chaque nouvelle itération, un graphe dont les indices ont été permutés. Parmi les partitions trouvées pendant l'itération, celle respectant la balance de partitionnement et de coût de coupe minimal est retournée. Si nous avons limité le temps d'exécution à 30 minutes, c'est qu'au delà il n'y a pas d'amélioration du coût de coupe. En effet, pendant ces 30 minutes, le logiciel est exécuté au moins une dizaine de milliers de fois, ce qui donne déjà une borne inférieure assez fiable du coût de coupe atteignable par celui-ci.

Le tableau 7.2 présente les résultats obtenus grâce à cette méthode itérative sur les logiciels pMetis et kMetis. Ils sont comparés avec les résultats de notre première implémentation de l'algorithme de fusion-fission, déjà utilisée dans la section précédente, et notée FF_1. Cependant, dans cette section l'algorithme FF_1 a été utilisé avec beaucoup plus de 1000 itérations. Le temps de calcul de la fusion-fission dépend de ce nombre d'itérations et de la taille des graphes, mais pour les résultats présentés dans ce tableau, nous avons réglé le nombre d'itérations pour que l'exécution dure entre 6 et 12 heures. Nous avons volontairement diminué le nombre d'itérations de manière inversement proportionnelle à la taille des graphes, afin de ne pas avoir des temps extrêmement longs.

La première partie de ce tableau présente les résultats de l'itération de pMetis avec ceux de FF_1, pour la balance de partitionnement native de pMetis : $1,01$. FF_1 utilise comme méthode de partitionnement local le logiciel pMetis (*supra* 4.4). Ainsi, cette première partie du tableau permet d'évaluer l'apport de la fusion-fission comme méthode globale utilisant une méthode « locale », en comparaison avec le meilleur résultat que celle-ci peut obtenir toute seule. Les meilleurs coûts de coupe sont écrits en gras, et lorsque deux coûts de coupe égaux ont été trouvés, ils sont écrits en italique. Nous pouvons observer que les résultats de la fusion-fission sont meilleurs que ceux de pMetis pour des partitions en 2 et 4 parties. Ils sont même tous au moins aussi bons dans ce dernier cas. Cependant, la situation s'inverse pour $k = 8$, bien que FF_1 ait encore trouvé un tiers de partitions de coût de coupe strictement inférieur.

La second partie de ce tableau rassemble les coûts de coupe obtenus avec kMetis et FF_1, pour la balance native de kMetis : $1,03$. Les résultats sont assez semblables à ceux de la comparaison précédente, à savoir que FF_1 obtient des résultats meilleurs que kMetis pour $k = 2$ et 4, mais des résultats un peu moins bons pour $k = 8$.

Les différents tests que nous avons faits nous ont montré qu'à partir de $k = 8$, les résultats trouvés avec la fusion-fission devenaient moins bons que pour $k = 2$ ou 4. Nous pensons que cela a pour origine deux problèmes :

– le premier est l'utilisation d'un algorithme d'affinage sur une partition déjà mal balancée. Nous avons constaté que plus le nombre k était grand, moins l'algorithme GKLR arrivait à affiner correctement les partitions. Pour corriger ce problème, nous avons utilisé à la place de cet algorithme la partie du code source du logiciel kMetis qui intégrait aussi une méthode de répartition de charge. L'algorithme FF_1 utilise cette partie de code source. Cependant, cette modification n'était toujours pas satisfaisante, car, aussi surprenant que cela soit, la méthode de répartition de charge accentuait régulièrement le déséquilibre des partitions au lieu de l'améliorer dans le cas où k était grand. Nous avons donc testé d'autres méthodes, comme nous le verrons dans les prochaines sections ;
– le second problème vient de l'algorithme de fusion-fission lui-même. Plus k est grand et plus les partitions sont éclatées à chaque itération. En effet, au début d'une itération sur la partition P où $card(P) = k'$, cherchant à créer une k''-partition, les parties de P sont divisées en k'' morceaux. Le nombre total de ces morceaux est donc $k' * k''$. Ce qui revient donc à créer une partition en $k' * k''$ parties, et cette opération donne de moins bons résultats quand k augmente. Pour contrer ce problème, nous avons proposé une adaptation de la séquence des nombres de

parties au cas où le carré du nombre maximal de parties d'une partition est de la même taille que le nombre de sommets (*supra* 4.4.6). Cependant, le choix du pourcentage de la valeur du nombre de sommets à prendre en compte a été empirique, et une valeur beaucoup plus faible pourrait être employée à la place. Il s'agit d'une voie que nous n'avons pas encore explorée, et qui pourrait permettre de résoudre ce défaut.

Nous devons aussi faire observer, concernant les résultats de cette section, que si les logiciels pMetis et kMetis ont été utilisés au moins une dizaine de milliers de fois sur chaque graphe, FF_1 n'a été exécuté qu'une seule fois.

L'une des spécificités de la méthode de fusion-fission est son aptitude à trouver en une exécution des partitions de cardinaux différents. La figure 7.4 permet d'évaluer la qualité des partitions trouvées en une exécution de la fusion-fission, en les comparant aux partitions trouvées après 30 minutes d'itérations de pMetis. Nous avons sélectionné deux graphes pour lesquels les résultats de la fusion-fission sont très différents pour $k = 8$: pour le premier, *bcsstk29* , la fusion-fission a trouvé un coût de coupe inférieur à celui de l'itération sur pMetis ; pour le second, *crack* , la fusion-fission a trouvé un coût de coupe supérieur à celui de l'itération sur pMetis. Cette figure montre une assez grande stabilité des coûts de coupes des partitions trouvées par la fusion-fission en comparaison de ceux trouvés par l'itération sur pMetis, et ce, même pour des partitions de cardinaux distants de $k = 8$. Cependant, sur l'extrémité droite de la figure, à partir des partitions en 14 ou 15 parties, les résultats trouvés par la fusion-fission semblent se détériorer un peu, mais cela reste marginal. Cette figure montre qu'en une exécution de la fusion-fission, celle-ci a permis de trouver des partitions de coût de coupe semblables (à plus ou moins 3% près dans la majorité des cas), à ceux de l'itération sur pMetis. Cette particularité de la fusion-fission peut servir dans le cas où un problème nécessite simultanément des partitions en un nombre différent de parties. Cependant, ce genre de problème est assez rare.

7.3 Améliorations de l'algorithme de fusion-fission

Cette section a pour objet de comparer les différentes « améliorations » de l'algorithme de fusion-fission proposées à la sous-section 4.4.6. Pour cela nous avons distingué différentes implémentations de l'algorithme de fusion-fission :

- **FF_2** est l'implémentation qui est la plus proche de FF_1 (d'où son nom). Afin de pouvoir être utilisée pour chercher une partition dont le nombre de parties est grand ($k > 16$), elle utilise l'adaptation de la séquence des nombres de parties au cas où le carré du nombre maximal de parties d'une partition est de la même taille que le nombre de sommets. Son critère d'arrêt est un temps d'exécution au lieu d'un nombre d'itérations comme c'est le cas pour FF_1. Cette implémentation va servir de base aux modifications des autres versions. Ainsi, ces dernières posséderont les deux améliorations dont nous venons de parler. Il y a deux grandes différences entre FF_2 et les autres implémentations que nous allons voir : la première est l'utilisation de pMetis comme algorithme de partitionnement ; la seconde concerne l'utilisation de l'algorithme GKLR du logiciel kMetis comme méthode d'affinage de la partition ;
- **FFraf** se différencie de FF_2 uniquement parce qu'il utilise en plus de l'algorithme d'affinage GKLR la méthode de répartition de charge présentée en section 2.7.2 ;
- **kFF** utilise le logiciel de partitionnement kMetis à la place de pMetis. Il utilise cependant lui aussi l'algorithme d'affinage GKLR ;
- **kFFraf** ajoute à l'implémentation de kFF la même méthode de répartition de charge que celle utilisée par FFraf ;
- **FFwc** est basé sur l'implémentation de FF_2 (elle utilise donc pMetis), mais utilise l'algorithme

		$bal = 1,00$					$bal = 1,01$					$bal = 1,03$				
	Graphe	FF$_2$	FFraf	kFF	kFFraf	FFwc	FF$_2$	FFraf	kFF	kFFraf	FFwc	FF$_2$	FFraf	kFF	kFFraf	FFwc
$k = 2$	add20	677	**631**	686	682	663	668	**629**	677	677	652	645	**571**	649	649	636
	data	**189**	**189**	190	**189**	**189**	**188**	**188**	**188**	**188**	**188**	**185**	**185**	186	186	**185**
	3elt	**90**	**90**	**90**	**90**	**90**	**89**	**89**	**89**	**89**	**89**	**87**	**87**	**87**	**87**	**87**
	uk	**20**	**20**	**20**	**20**	**20**	**19**	**19**	**19**	**19**	**19**	19	19	**18**	**18**	19
	add32	**11**	**11**	**11**	**11**	**11**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
	bcsstk33	**10174**	**10174**	err	err	***10171***	**10097**	**10097**	err	err	**10097**	**10064**	10097	err	err	**10064**
	whitaker3	**127**	**127**	**127**	**127**	**127**	**126**	**126**	**126**	**126**	**126**	**126**	**126**	**126**	**126**	**126**
	crack	**184**	**184**	**184**	**184**	**184**	**183**	**183**	184	184	**183**	**182**	**182**	184	184	**182**
	wing_nodal	1740	**1730**	1743	1732	***1713***	1725	1726	**1717**	1724	***1704***	**1703**	1708	1706	1706	***1692***
	fe_4elt2	**130**	**130**	**130**	**130**	**130**	**130**	**130**	**130**	**130**	**130**	**130**	**130**	**130**	**130**	**130**
	vibrobox	11548	**11543**	11728	11729	***11286***	**11516**	**11516**	11574	11574	***11262***	**11481**	**11481**	11573	11573	***11184***
	bcsstk29	**2843**	**2843**	2846	2851	**2843**	**2818**	**2818**	**2818**	2849	**2818**	**2818**	**2818**	**2818**	2849	**2818**
	4elt	**139**	140	140	140	**139**	**138**	**138**	**138**	**138**	**138**	**137**	**137**	138	138	**137**
	fe_sphere	**386**	**386**	406	406	**386**	**384**	386	388	406	386	**384**	386	388	406	**384**
	cti	**334**	**334**	**334**	**334**	**334**	**318**	**318**	**318**	**318**	**318**	**318**	**318**	**318**	**318**	**318**
	memplus	**6065**	**6065**	6299	6261	***5733***	**5973**	5986	6121	6133	***5630***	**5866**	5869	5991	5969	***5479***
$k = 4$	add20	**1205**	1208	1208	1209	1214	1202	1208	**1198**	**1198**	***1194***	**1194**	1196	1195	1195	***1181***
	data	**390**	392	394	392	393	**381**	384	385	386	***380***	**380**	381	384	384	***375***
	3elt	**205**	**205**	214	212	***201***	**201**	203	202	202	***199***	**199**	200	**199**	**199**	**199**
	uk	**45**	**45**	51	54	46	**44**	45	**44**	**44**	**44**	43	43	**42**	**42**	44
	add32	**34**	**34**	**34**	35	***34***	**33**	**33**	**33**	**33**	**33**	**33**	**33**	**33**	**33**	**33**
	bcsstk33	**22136**	**22136**	err	err	***21799***	22115	**22059**	err	err	***21547***	**21393**	21647	err	err	***20886***
	whitaker3	**389**	**389**	405	398	***383***	**382**	385	388	391	**382**	**382**	383	385	389	***381***
	crack	**373**	**373**	382	388	374	**370**	**370**	374	380	***364***	**366**	370	373	377	***361***
	wing_nodal	**3733**	**3733**	3787	3780	***3651***	**3684**	**3684**	3703	3714	***3609***	**3664**	3684	3688	3702	***3606***
	fe_4elt2	351	**349**	355	353	350	**349**	**349**	351	351	**349**	**349**	**349**	351	351	***346***
	vibrobox	**19812**	**19812**	20222	20257	***19388***	**19637**	19792	19752	19791	***19350***	19615	19702	**19466**	**19466**	***19122***
	bcsstk29	**8693**	**8693**	err	9043	***8232***	**8463**	8659	err	9043	***8194***	**8322**	8578	err	8817	***8194***
	4elt	**340**	**340**	352	345	***340***	336	340	335	**332**	***324***	332	334	**331**	332	***320***
	fe_sphere	811	**809**	846	844	***773***	811	**802**	814	820	***768***	808	800	**799**	820	***768***
	cti	**990**	**990**	1069	1080	***990***	970	**968**	981	1031	***946***	**942**	952	974	985	***917***
	memplus	**9891**	**9891**	10496	10215	***9849***	**9865**	**9865**	10145	10145	***9809***	9863	**9844**	9952	9994	***9676***

TAB. 7.3 – Comparaison des différentes versions de l'algorithme de fusion-fission pour $k = 2$ et 4.

		$bal = 1,00$					$bal = 1,01$					$bal = 1,03$				
	Graphe	FF_2	FFraf	kFF	kFFraf	FFwc	FF_2	FFraf	kFF	kFFraf	FFwc	FF_2	FFraf	kFF	kFFraf	FFwc
$k = 8$	add20	1860	1895	1825	**1820**	***1757***	1806	1806	**1775**	1786	***1744***	1760	1760	**1752**	1776	***1721***
	data	**716**	724	732	748	***699***	693	700	**687**	713	***680***	693	**674**	687	689	***658***
	3elt	**378**	380	399	399	***366***	**353**	357	355	359	***349***	**346**	349	349	352	***339***
	uk	**102**	**102**	184	maxint	101	**95**	96	100	100	98	95	96	**94**	95	97
	add32	**97**	**97**	156	151	102	**81**	**81**	83	87	***81***	80	81	**77**	78	***74***
	bcsstk33	**36016**	**36016**	37689	37924	***35265***	36016	36016	36159	**35786**	***34488***	35171	35703	**35122**	35673	***34200***
	whitaker3	719	**713**	746	754	***680***	684	**683**	687	693	***664***	683	683	**682**	687	***661***
	crack	**750**	757	824	812	***726***	**711**	721	716	734	***691***	**705**	717	712	725	***685***
	wing_nodal	**5915**	**5915**	6025	6211	***5723***	**5737**	5888	5882	5813	***5683***	**5737**	5746	5783	5813	***5604***
	fe_4elt2	**648**	654	650	655	***635***	**631**	634	639	646	***621***	**629**	634	630	639	***617***
	vibrobox	26626	**26507**	27262	28068	***25425***	25729	25883	**25702**	26178	***25066***	25420	**25367**	25586	25586	***24701***
	bcsstk29	16790	**16784**	17783	17872	***15459***	**15194**	15356	15820	15938	***14926***	**15071**	15264	15541	15816	***14659***
	4elt	**593**	605	605	631	600	**572**	578	582	589	***550***	**560**	565	561	581	***533***
	fe_sphere	**1311**	1324	1342	1395	***1255***	1284	1281	1282	**1274**	***1160***	1276	1277	1279	**1274**	***1152***
	cti	**1913**	2069	2476	2475	***1973***	**1913**	1948	2056	2108	***1854***	**1913**	1948	2055	2108	***1800***
	memplus	**13110**	**13110**	13507	13187	***12599***	**12594**	13110	12974	13007	***12552***	12594	13110	**12565**	12679	***12446***
$k = 16$	add20	maxint	**2283**	2371	2397	***2159***	2333	**2235**	2371	2397	***2142***	2244	**2177**	2235	2397	***2107***
	data	**1249**	1274	1258	1265	***1180***	**1184**	1203	1199	1201	***1180***	**1160**	1190	1167	1181	***1146***
	3elt	**665**	**665**	716	719	***612***	**598**	601	599	602	***588***	**583**	593	588	585	***575***
	uk	197	maxint	**188**	211	***186***	**183**	189	**183**	192	***177***	**177**	182	178	183	***171***
	add32	**128**	**128**	248	281	**128**	**128**	**128**	149	174	**128**	**128**	**128**	146	157	**128**
	bcsstk33	**59791**	**59791**	61930	63286	***57273***	58049	58196	**57668**	58032	***56727***	57130	57854	**56961**	57425	***56366***
	whitaker3	**1221**	1237	1224	1230	***1163***	**1165**	1174	1171	1169	***1137***	1156	1167	**1150**	1169	***1120***
	crack	**1255**	**1255**	1400	1408	***1199***	1176	1189	**1161**	1175	***1141***	1165	1177	**1161**	1165	***1130***
	wing_nodal	9282	**9222**	9231	9358	***8822***	**8946**	9222	9150	8993	***8733***	**8755**	8977	8761	8993	***8652***
	fe_4elt2	1093	1152	**1092**	1134	***1057***	**1061**	1068	1066	1064	***1048***	**1050**	1060	1054	1060	***1032***
	vibrobox	37441	37441	**37329**	38031	***35575***	36781	37441	**36418**	38031	***35132***	35345	35649	**35225**	**35225**	***34793***
	bcsstk29	28615	28667	**28273**	28478	***25626***	28151	27131	**26475**	28236	***24705***	**24781**	25026	24972	25455	***24147***
	4elt	**1056**	**1056**	1122	1194	***1049***	**1040**	1056	1054	1065	***990***	**994**	1027	1010	1044	***947***
	fe_sphere	**2030**	**2030**	2080	2163	***1902***	**1933**	**1933**	1936	1952	***1745***	**1923**	**1923**	1927	1930	***1716***
	cti	**3181**	**3181**	3850	3911	**3181**	**3181**	**3181**	3289	3488	***3082***	**3181**	**3181**	3289	3313	***3018***
	memplus	maxint	maxint	16522	**15681**	***14725***	**14931**	14942	14985	15342	***14590***	14800	14942	**14785**	15086	***14507***

TAB. 7.4 – Comparaison des différentes versions de l'algorithme de fusion-fission pour $k = 8$ et 16.

		$bal = 1,00$					$bal = 1,01$					$bal = 1,03$				
	Graphe	FF_2	FFraf	kFF	kFFraf	FFwc	FF_2	FFraf	kFF	kFFraf	FFwc	FF_2	FFraf	kFF	kFFraf	FFwc
$k = 32$	add20	maxint	**2823**	3263	3241	***2681***	maxint	**2823**	3263	3241	***2681***	2798	**2696**	3003	3241	***2570***
	data	**1955**	1966	1995	2002	***1912***	**1955**	1966	1995	2002	***1912***	**1909**	1914	1943	1919	***1889***
	3elt	1047	**1034**	1046	1044	***1011***	1014	**1010**	1012	1017	1011	999	1001	**996**	1007	***987***
	uk	**311**	maxint	329	maxint	***302***	**309**	331	329	333	***286***	287	291	**285**	296	***284***
	add32	maxint	maxint	maxint	maxint	***223***	**213**	214	**213**	224	***212***	**213**	214	**213**	224	***212***
	bcsstk33	**86008**	**86008**	91313	92409	***81796***	**81590**	82645	81638	82557	***81327***	81124	81862	**80990**	81534	***80112***
	whitaker3	**1847**	1857	1860	1876	***1771***	1773	1785	**1768**	1778	***1738***	1761	1770	**1756**	1762	***1724***
	crack	**1890**	**1890**	2250	maxint	***1878***	**1815**	1837	1826	1843	***1772***	**1801**	1824	1805	1822	***1745***
	wing_nodal	13237	**13086**	14254	14272	***12667***	**12770**	13086	12886	14272	***12481***	**12512**	13086	12513	14272	***12426***
	fe_4elt2	**1787**	**1787**	1798	1849	***1710***	**1720**	1732	1728	1735	***1689***	**1709**	1720	1718	1726	***1680***
	vibrobox	**46112**	**46112**	46822	46943	***44287***	**46112**	**46112**	46119	46943	***43772***	43566	44052	**43190**	43862	43321
	bcsstk29	**41190**	**41190**	44241	45858	***39436***	**39246**	39908	40198	40645	***38657***	**38406**	39140	38895	40087	***38031***
	4elt	**1769**	**1769**	1800	1913	***1721***	1737	1769	**1725**	1747	***1639***	**1667**	1677	1684	1709	***1610***
	fe_sphere	**2913**	**2913**	2986	3045	***2713***	2772	2789	**2770**	2789	***2575***	2759	2788	**2756**	2778	***2507***
	cti	**4605**	**4605**	5407	5537	**4605**	**4605**	**4605**	4685	4744	***4515***	**4605**	**4605**	4685	4744	***4403***
	memplus	maxint	maxint	17671	**17524**	***16051***	17303	**16155**	16219	17524	***15789***	16205	**16155**	16219	17524	***15702***
$k = 64$	add20	maxint	**3351**	3730	3691	***3164***	maxint	**3351**	3730	3691	***3164***	3488	**3213**	3661	3618	***3093***
	data	**3144**	3161	3192	3168	***3028***	**3144**	3161	3192	3168	***3028***	**2966**	2989	2985	3008	***2918***
	3elt	**1667**	1690	1723	1718	***1648***	**1667**	1690	1723	1718	***1648***	**1597**	1603	1607	1614	***1596***
	uk	**488**	503	510	516	***474***	**488**	503	510	516	***474***	456	**455**	476	476	462
	add32	maxint	maxint	537	**528**	***513***	maxint	maxint	537	**528**	***513***	516	527	**502**	507	503
	bcsstk33	**116288**	116963	117153	116476	***112932***	113504	113397	**113075**	113173	***112932***	112475	112494	**112156**	112256	***111219***
	whitaker3	2779	**2777**	2828	2851	***2651***	**2678**	2680	2693	2692	***2651***	2650	2650	**2644**	2659	***2602***
	crack	maxint	maxint	maxint	maxint	***2806***	**2734**	2738	2791	2817	***2691***	**2693**	2696	2746	2758	***2650***
	wing_nodal	19401	**18829**	19371	19524	***17124***	**17899**	**17899**	19371	19524	***16909***	16795	17899	**16773**	19524	***16700***
	fe_4elt2	**2733**	2743	2837	2877	***2639***	**2635**	2653	2687	2694	2639	**2635**	2637	2668	2668	***2605***
	vibrobox	54990	54930	**54343**	54486	***51270***	**53764**	**53764**	54343	54486	***50878***	51473	53764	**50502**	54486	***50105***
	bcsstk29	65172	**64892**	66944	68523	***60883***	**60225**	61292	62469	61591	***60204***	**60225**	61292	62469	61591	***59119***
	4elt	**2871**	2953	3041	3738	***2815***	**2836**	2953	2891	3119	***2725***	**2757**	2854	2773	2909	***2682***
	fe_sphere	4191	4191	**4162**	4225	***3840***	**3932**	3937	3943	3936	***3738***	3920	3925	**3878**	3931	***3644***
	cti	**6461**	**6461**	7214	7365	***6415***	6377	**6376**	6486	6515	***6365***	6377	**6376**	6486	6515	***6245***
	memplus	18886	18397	18267	**18238**	***17859***	18886	**18083**	18267	18238	***17562***	17894	**17735**	17764	17896	***17361***

TAB. 7.5 – Comparaison des différentes versions de l'algorithme de fusion-fission pour $k = 32$ et 64.

d'affinage et de répartition de charge de Walshaw-Cross.

Toutes ces implémentations sont auto-suffisantes. Il n'est pas nécessaire de les paramétrer d'une façon spéciale pour les utiliser. Elles ne nécessitent que la connaissance du graphe, du nombre de parties de la partition cherchée et du temps d'exécution minimum (l'algorithme cherche déjà des partitions pour 4 balances de partitionnement différentes : 1,00 ; 1,01 ; 1,03 et 1,05).

Les tableaux 7.3 à 7.5 présentent les coûts de coupe de partitions trouvées par ces différentes versions de l'algorithme de fusion-fission. Les résultats trouvés par FFwc sont placés séparément, car ils ont été trouvés après 4 heures de calculs, alors que les autres implémentations n'ont bénéficié « que » de 2 heures de calculs.

Mise à part le fait que FFwc ait eu deux fois plus de temps de calcul que les autres versions de l'algorithme de fusion-fission, c'est de loin celle qui permet de trouver les coûts de coupe les plus bas. Lorsque FFwc trouve un coût de coupe plus petit que ceux des 4 autres versions, celui-ci est écrit en gras et en italique, et lorsqu'il est égal à la valeur la plus basse trouvée par les autres versions, il est écrit en gras.

Si l'on ne considère pas les résultats de FFwc, alors nous pouvons faire les constats suivants :

- d'une manière générale, les implémentations qui utilisent pMetis sont plus performantes que celles utilisant kMetis. Cependant, la différence diminue un peu quand la balance de partitionnement augmente ;
- dans le cas de FF et FFraf, si l'algorithme de répartition de charge permet de trouver dans certains cas des partitions de coût de coupe plus bas que sans lui, son utilisation ralentit considérablement l'algorithme de fusion-fission qui ne va pas donc pas avoir la possibilité d'explorer l'espace de recherche aussi bien que sans cet algorithme. L'apport de cet algorithme de répartition de charge est donc en demi-teinte : d'un côté de nouvelles solutions sont trouvées qui n'avaient pu être envisagées avant à cause des défaillances de l'algorithme GKLR, d'un autre côté, des solutions n'ont pu être atteintes dans le même temps de calcul à cause de la charge supplémentaire que représentait cet algorithme. Ainsi, pour vraiment profiter de cette répartition de charge, il faut accorder un temps de calcul supplémentaire à l'algorithme ;
- le logiciel kMetis est beaucoup moins stable que pMetis. En effet, il lui arrive de trouver des partitions en un nombre de parties différent de celui qui lui a été demandé. Les partitions pour lesquelles ce problème est survenu ont, à la place de leur coût de coupe : *err*, pour signaler ce problème ;
- la différence entre les logiciels pMetis et kMetis réside principalement dans leur algorithme d'affinage. Le premier ne peut affiner qu'une bissection, alors que le second peut affiner une partition en un nombre de parties quelconque. Les moins bons résultats obtenus par kFF par rapport à ceux de FF_2 nous font nous demander si nous avons bien fait d'utiliser comme algorithme d'affinage, GKLR. Cette impression est renforcée par les bons résultats obtenus par FFwc.

7.4 Meilleurs résultats

Cette section compare les meilleurs résultats trouvés par nos différentes implémentations de l'algorithme de fusion-fission, et les partitions des archives du partitionnement de graphe de Chris Walshaw (*infra* annexe A.4). Comme nous l'avons dit, celle-ci regroupe les meilleurs partitions en $k = 2, 4, 8, 16, 32$ et 64 parties, trouvées par 20 outils de partitionnements, pour des balances de partitionnement de : $1,00$; $1,01$; $1,03$ et $1,05$.

Les tableaux 7.8 à 7.13 présentent ces comparaisons. Nous avons utilisé le même format de ta-

Valeur de k	Nb. occurrences $< coupe\_min$	Nb. occurrences $\leq coupe\_min$
2	5 (4%)	62 (50%)
4	56 (45%)	76 (61%)
8	33 (27%)	41 (33%)
16	12 (10%)	12 (10%)
32	10 (8%)	16 (13%)
64	20 (16%)	21 (17%)

TAB. 7.6 – Récapitulatif du nombre de fois où une implémentation de la fusion-fission a trouvé une partition de qualité soit meilleure, soit au moins égale à celle des archives du partitionnement de graphe.

Abréviation	Logiciel
MRSB	Bissection récursive spectrale et multi-niveaux [BAR94]
Ch2.0	Chaco utilisé avec sa méthode de bissection récursive multi-niveaux, version 2.0 (octobre 1995, *supra* A.1)
pM4.0	pMetis, version 4.0 (septembre 1998, *supra* A.1)
J2.2	Jostle, version 2.2 (mars 2000, *supra* A.1)
iJ	Recherche itérative locale utilisant Jostle [WAL04]
JE	Jostle évolutionnaire (*supra* 3.5.2)
GrPart	Recherche itérative locale utilisant l'algorithme multi-niveaux d'Alexander Kozhushkin
MLSAT	Méthode multi-niveaux utilisant comme méthode d'affinage un algorithme hybride de recuit simulé et de recherche tabou [BAÑ03]
AMG	Algorithme de bissection utilisant l'algèbre *MultiGrid* [RON05]
MQI	Méthode itérative de bissection utilisant pMetis et un algorithme de flux-maximum [LAN04b]
SDP	Approche semi-définie utilisant un algorithme de flux-maximum de Lang-Rao
mpM4.0	Recherche itérative locale utilisant pMetis
JKM	Heuristique pour le partitionnement de graphe de Jan Kasper Martinsen et Xing Cai
GCSVD	Algorithme évolutionnaire de Jacob Martin [MAR05]
Probe	Méthode Probe de P. Chardaire, M. Barake, et G. McKeown [CHA07]
N/A	Source inconnue (probablement Jostle évolutionnaire)
FF_1	Première implémentation de l'algorithme de fusion-fission
FF_2	Seconde implémentation de l'algorithme de fusion-fission, utilise pMetis et l'algorithme GKLR du logiciel kMetis
FFraf	Algorithme FF_2 utilisant la répartition de charge présentée en section 2.7.2
kFF	Algorithme FF_2 utilisant kMetis au lieu de pMetis
kFFraf	Algorithme kFF utilisant la répartition de charge présentée en section 2.7.2
FFwc	Algorithme FF_2 utilisant l'algorithme d'affinage et de répartition de charge de Walshaw-Cross à la place de celui de kMetis
FF*	désigne FF_1, FF_2, FFraf, kFF, kFFraf et FFwc
kFF*	désigne kFF et kFFraf
FF*-kFF	désigne toutes les implémentations de FF*, sauf kFF

TAB. 7.7 – Tableau de correspondance entre les acronymes et les noms des logiciels

$bal = 1,00$	$k = 2$				$k = 4$				$k = 8$			
Graphe	Algo.	Coupe	Coupe	FF	Algo.	Coupe	Coupe	FF	Algo.	Coupe	Coupe	FF
add20	Probe	**596**	631	FFraf	JE	**1203**	1205	FF_2	iJ	1758	**1757**	FFwc
data	Probe	*189*	*189*	FF*-kFF	iJ	429	**390**	FF_2	iJ	728	**699**	FFwc
3elt	JE	*90*	*90*	FF*	JE	*201*	*201*	FFwc	JE	**349**	366	FFwc
uk	mpM4.0	*20*	*20*	FF*	JE	*44*	*44*	FF_1	JE	**91**	99	FF_2
add32	Ch2.0	*11*	*11*	FF*	pM4.0	37	**34**	FF*-kFFraf	JE	75	**72**	FF_2
bcsstk33	GCSVD	*10171*	*10171*	FF, FFwc	iJ	21956	**21799**	FFwc	iJ	**35088**	35265	FFwc
whitaker3	JE	*127*	*127*	FF*	JE	**382**	383	FFwc	JE	**664**	680	FFwc
crack	JE	*184*	*184*	FF*	JE	**368**	370	FF_1	JE	**687**	721	FF_1
wing_nodal	JE	**1707**	1713	FFwc	JE	**3581**	3651	FFwc	JE	**5443**	5723	FFwc
fe_4elt2	MRSB	*130*	*130*	FF*	JE	*349*	*349*	FFraf	JE	**617**	635	FFwc
vibrobox	JE	**10343**	11286	FFwc	JE	**19245**	19388	FFwc	iJ	25468	**25425**	FFwc
bcsstk29	mpM4.0	*2843*	*2843*	FF*-kFF*	MRSB	8786	**8232**	FFwc	iJ	16832	**15459**	FFwc
4elt	JE	*139*	*139*	FF, FFwc	JE	**327**	338	FF_1	JE	**556**	593	FF_2
fe_sphere	JE	*386*	*386*	FF*-kFF*	N/A	*773*	*773*	FFwc	JE	**1193**	1255	FFwc
cti	JE	*334*	*334*	FF*	JE	977	**963**	FF_1	JE	**1812**	1913	FF_2
memplus	Probe	**5513**	5721	FF_1	JE	**9754**	9819	FF_1	iJ	**12095**	12599	FFwc
cs4	JE	**372**	411	FF_1	JE	**964**	1053	FF_2, FFraf	JE	**1496**	1746	FF_1, FF_2, FFraf
bcsstk30	JE	*6394*	*6394*	FF*-kFF*	JE	**16652**	16837	FF_1	JE	**34921**	35681	FFwc
bcsstk31	Probe	**2762**	2768	FFwc	iJ	8322	**7593**	FFwc	pM4.0	14426	**14216**	FFwc
bcsstk32	JE	**4667**	4739	FFraf	JE	10578	**9820**	FFwc	JE	**22826**	23537	FF_1
t60k	JE	**80**	84	FF_2	JE	**216**	225	FFwc	JE	**480**	508	FFwc
wing	MQI	**791**	885	FF_2, FFraf	JE	**1666**	1846	FF_2	JE	**2589**	3018	FF_1
brack2	JE	*731*	*731*	FF_1, FFwc, kFFraf	JE	**3090**	3110	FFwc	JE	**7269**	7730	FFwc
finan512	Ch2.0	*162*	*162*	FF*	Ch2.0	*324*	*324*	FF*	Ch2.0	*648*	*648*	FF_2, FFwc
fe_tooth	mpM4.0	**3850**	3865	FFwc	iJ	7605	**7289**	FFwc	iJ	**12850**	12951	FFwc
fe_rotor	Probe	**2098**	2116	kFF	Ch2.0	8097	**7845**	FF_1	Ch2.0	14953	**14851**	FFwc
598a	Probe	**2398**	2434	kFFraf	Ch2.0	8379	**8277**	FF_1	Ch2.0	17223	**17165**	FFwc
fe_ocean	Probe	**464**	468	FF_1	JE	**1923**	1944	FF_1	N/A	4760	**4516**	FF_2, FFraf, FFwc
144	Probe	**6490**	6663	FFwc	N/A	**15964**	16215	FF_1	N/A	**27751**	28254	FF*-kFF*
wave	Probe	**8692**	8874	FFwc	iJ	18774	**18070**	FF_1	iJ	**32204**	32270	FF_1
m14b	MQI	**3836**	3890	FFwc	Ch2.0	14013	**13508**	FFwc	pM4.0	**28128**	28453	FFwc
$< coupe\_min$			0	0%			12	39%			9	29%
$\leq coupe\_min$			15	48%			17	55%			10	32%

TAB. 7.8 – Meilleures partitions trouvées par l'algorithme de fusion-fission comparées aux résultats de la « Graph Partitioning Archive », pour une balance de partitionnement $balance = 1,00$

$bal = 1,00$	$k = 16$				$k = 32$				$k = 64$			
Graphe	Algo.	Coupe	Coupe	FF	Algo.	Coupe	Coupe	FF	Algo.	Coupe	Coupe	FF
add20	iJ	2216	**2159**	FFwc	JE	2765	**2681**	FFwc	JE	3266	**3164**	FFwc
data	Ch2.0	1245	**1180**	FFwc	iJ	2004	**1912**	FFwc	iJ	**3016**	3028	FFwc
3elt	JE	**589**	612	FFwc	JE	**972**	1011	FFwc	iJ	**1594**	1648	FFwc
uk	JE	**162**	186	FFwc	JE	**286**	302	FFwc	iJ	**456**	474	FFwc
add32	Ch2.0	**121**	128	FF_2, FFraf, FFwc	JE	234	**223**	FFwc	Ch2.0	720	**513**	FFwc
bcsstk33	iJ	**56973**	57273	FFwc	iJ	**80099**	81796	FFwc	iJ	**109585**	112932	FFwc
whitaker3	JE	**1110**	1163	FFwc	JE	**1719**	1771	FFwc	iJ	**2579**	2651	FFwc
crack	JE	**1124**	1199	FFwc	JE	**1768**	1878	FFwc	iJ	**2696**	2806	FFwc
wing_nodal	JE	**8422**	8822	FFwc	iJ	**12228**	12667	FFwc	iJ	**16373**	17124	FFwc
fe_4elt2	JE	**1028**	1057	FFwc	JE	**1677**	1710	FFwc	iJ	**2537**	2639	FFwc
vibrobox	iJ	**33468**	35575	FFwc	JE	**43786**	44287	FFwc	JE	**51101**	51270	FFwc
bcsstk29	iJ	**24706**	25626	FFwc	iJ	**38535**	39436	FFwc	iJ	61252	**60883**	FFwc
4elt	JE	**968**	1049	FFwc	JE	**1606**	1721	FFwc	iJ	**2702**	2815	FFwc
fe_sphere	JE	**1750**	1902	FFwc	JE	**2567**	2713	FFwc	JE	**3663**	3840	FFwc
cti	JE	**2915**	3181	FF_2, FFraf, FFwc	iJ	**4460**	4605	FF_2, FFraf, FFwc	iJ	**6277**	6415	FFwc
memplus	JE	**13742**	14725	FFwc	iJ	**14830**	16051	FFwc	JE	**17446**	17859	FFwc
cs4	JE	**2206**	2538	FF_2, FFraf	iJ	**3110**	3579	FF_2, FFraf	iJ	**4223**	4762	kFF
bcsstk30	JE	**72007**	76852	FFwc	JE	**120580**	125072	FFwc	JE	**184855**	186397	FFwc
bcsstk31	iJ	**25491**	27126	FFwc	iJ	**39440**	41741	FFwc	iJ	63998	**63609**	FFwc
bcsstk32	JE	**40275**	43371	FF_2, FFraf, FFwc	JE	**66407**	70020	FF_2, FFraf, FFwc	JE	**102820**	104721	FFwc
t60k	iJ	**892**	985	FFwc	iJ	**1453**	1613	FF_2, FFraf, FFwc	iJ	**2277**	2484	FF_2, FFraf, FFwc
wing	JE	**4198**	4666	FF_2, FFraf, FFwc	iJ	**6104**	6700	FF_2, FFraf, FFwc	iJ	**8203**	9235	FFwc
brack2	JE	**12324**	12655	FF_2, FFraf, FFwc	iJ	**18421**	19786	FF_2, FFraf, FFwc	iJ	**27881**	29011	FFwc
finan512	Ch2.0	**1296**	2656	FFwc	Ch2.0	**2592**	4576	FFwc	pM4.0	**11192**	12705	FFwc
fe_tooth	iJ	**18804**	19346	FF_2, FFraf, FFwc	iJ	**26642**	28299	FFwc	iJ	**36766**	39131	FFwc
fe_rotor	iJ	**21241**	23863	FF_2, FFraf, FFwc	iJ	**34320**	35775	FFwc	pM4.0	52626	**51743**	FFwc
598a	iJ	**28152**	28309	FFwc	iJ	**42088**	43069	FFwc	iJ	**59708**	62424	FFwc
fe_ocean	iJ	**8622**	9339	FFwc	iJ	**14277**	14613	FF_2, FFraf, FFwc	iJ	**22301**	23121	FFwc
144	N/A	**39856**	42945	FFwc	iJ	**58659**	62242	FFwc	iJ	**82123**	86606	FFwc
wave	iJ	**47277**	48106	FF_2, FFraf, FFwc	iJ	**66891**	69426	FFwc	iJ	**91784**	94037	FFwc
m14b	iJ	**45362**	48866	FFwc	N/A	**68468**	74197	FFwc	iJ	**103946**	107844	FFwc
$<$ *coupe_min*			2	6%			3	10%			5	16%
$\leq$ *coupe_min*			2	6%			3	10%			5	16%

TAB. 7.9 – Meilleures partitions trouvées par l'algorithme de fusion-fission comparées aux résultats de la « Graph Partitioning Archive », pour une balance de partitionnement $balance = 1,00$

$bal = 1,01$	$k = 2$				$k = 4$				$k = 8$			
Graphe	Algo.	Coupe	Coupe	FF	Algo.	Coupe	Coupe	FF	Algo.	Coupe	Coupe	FF
add20	GCSVD	**594**	629	FFraf	JE	1203	**1194**	FFwc	iJ	1758	**1744**	FFwc
data	AMG	*188*	*188*	FF*	iJ	425	**380**	FFwc	iJ	719	**680**	FFwc
3elt	GrPart	*89*	*89*	FF*	JE	*199*	*199*	FF_1, FFwc	JE	*349*	*349*	FFwc
uk	AMG	*19*	*19*	FF*	JE	44	**43**	FF_1	JE	**86**	95	FF_2
add32	J2.2	*10*	*10*	FF*	JE	*33*	*33*	FF*	JE	69	**66**	FF_1
bcsstk33	GrPart	10109	**10097**	FF*	iJ	21685	**21547**	FFwc	iJ	35088	**34488**	FFwc
whitaker3	JE	*126*	*126*	FF*-kFF	JE	**380**	382	FF_2, FFwc	JE	**660**	664	FFwc
crack	AMG	*183*	*183*	FF*-kFF	JE	368	**364**	FFwc	N/A	**678**	691	FFwc
wing_nodal	MQI	**1696**	1704	FF_1, FFwc	JE	**3572**	3609	FFwc	JE	**5443**	5683	FFwc
fe_4elt2	MRSB	*130*	*130*	FF*	JE	*349*	*349*	FF*-kFF*	iJ	**611**	621	FFwc
vibrobox	JE	**10310**	11262	FF_1, FFwc	JE	**19245**	19350	FFwc	iJ	**25001**	25066	FFwc
bcsstk29	GrPart	*2818*	*2818*	FF*-kFFraf	MRSB	8786	**8194**	FFwc	iJ	16505	**14926**	FF_1, FFwc
4elt	GrPart	*138*	*138*	FF*	JE	327	**324**	FFwc	JE	556	**550**	FFwc
fe_sphere	JE	386	**384**	FF_2	N/A	*768*	*768*	FFwc	JE	**1152**	1160	FFwc
cti	JE	*318*	*318*	FF*	N/A	967	**946**	FFwc	JE	**1812**	1854	FFwc
memplus	JE	**5492**	5630	FFwc	JE	**9754**	9792	FF_1	iJ	**11939**	12552	FFwc
cs4	JE	**367**	411	FF_1	JE	**940**	1043	FF_1	N/A	**1476**	1693	FF_1
bcsstk30	GrPart	*6335*	*6335*	FF_1, FF_2, kFF	N/A	**16622**	16687	FF_1	JE	**34604**	35357	FF_2, FFraf
bcsstk31	GrPart	**2701**	2703	FF_1	pM4.0	7879	**7506**	FFwc	pM4.0	14426	**14010**	FFwc
bcsstk32	JE	**4667**	4739	FFraf	JE	10578	**9538**	FF_1	JE	**22826**	23537	FF_1
t60k	AMG	**77**	81	FFwc	iJ	**213**	222	FF_2	N/A	**467**	505	FFwc
wing	MQI	**787**	870	FF_2, FFraf	JE	**1666**	1840	FF_1	JE	**2589**	3001	FF_1
brack2	GrPart	**708**	711	FF_1, kFF	JE	3090	**3089**	FFwc	JE	**7269**	7571	FFwc
finan512	Ch2.0	*162*	*162*	FF*	Ch2.0	*324*	*324*	FF*	Ch2.0	*648*	*648*	FF*-FFraf
fe_tooth	SDP	**3823**	3858	FFwc	iJ	7435	**7234**	FFwc	iJ	12850	**12725**	FF_1
fe_rotor	GrPart	2049	**2045**	FF_1	Ch2.0	8097	**7667**	FF_2, FFraf	iJ	14028	**13786**	FF_1
598a	MQI	**2388**	2423	FF_1	N/A	8311	**8222**	FF_1	iJ	**16594**	17006	FF_1
fe_ocean	GrPart	**387**	391	FF_1, FF_2	N/A	1911	**1878**	FF_1	N/A	4760	**4538**	FF_2, FFraf, FFwc
144	MQI	**6479**	6655	FFwc	N/A	**15345**	16046	FF_1	N/A	**25818**	27872	FF_1
wave	MQI	**8682**	8848	FFwc	iJ	18058	**17950**	FF_1	iJ	32204	**31965**	FF_1
m14b	MQI	**3826**	3880	FFwc	JKM	13844	**13430**	FF_1	pM4.0	28128	**27815**	FF_1
$<$ *coupe_min*			3	10%			17	55%			12	39%
$\leq$ *coupe_min*			15	48%			22	71%			14	45%

TAB. 7.10 – Meilleures partitions trouvées par l'algorithme de fusion-fission comparées aux résultats de la « Graph Partitioning Archive », pour une balance de partitionnement $balance = 1,01$

$bal = 1,01$	$k = 16$				$k = 32$				$k = 64$			
Graphe	Algo.	Coupe	Coupe	FF	Algo.	Coupe	Coupe	FF	Algo.	Coupe	Coupe	FF
add20	iJ	2216	**2142**	FFwc	JE	2765	**2681**	FFwc	JE	3266	**3164**	FFwc
data	Ch2.0	1245	**1180**	FFwc	iJ	2004	**1912**	FFwc	iJ	**3016**	3028	FFwc
3elt	JE	589	**588**	FFwc	JE	**972**	1011	FFraf, FFwc	iJ	**1594**	1648	FFwc
uk	JE	**157**	177	FFwc	iJ	**285**	286	FFwc	iJ	**456**	474	FFwc
add32	JE	**117**	128	FF_2, FFraf, FFwc	JE	*212*	*212*	FFwc	Ch2.0	720	**513**	FFwc
bcsstk33	iJ	**55546**	56727	FFwc	iJ	**79121**	81327	FFwc	iJ	**109585**	112932	FFwc
whitaker3	JE	**1110**	1137	FFwc	JE	**1719**	1738	FFwc	iJ	**2579**	2651	FFwc
crack	JE	**1124**	1141	FFwc	iJ	**1707**	1772	FFwc	iJ	**2623**	2691	FFwc
wing_nodal	JE	**8422**	8733	FFwc	iJ	**12228**	12481	FFwc	iJ	**16221**	16909	FFwc
fe_4elt2	JE	**1028**	1048	FFwc	JE	**1677**	1689	FFwc	iJ	**2537**	2635	FF_2
vibrobox	iJ	**33468**	35132	FFwc	JE	**43415**	43772	FFwc	JE	**50849**	50878	FFwc
bcsstk29	iJ	24706	**24705**	FFwc	iJ	**37810**	38657	FFwc	iJ	60795	**60204**	FFwc
4elt	iJ	**961**	990	FFwc	JE	**1606**	1639	FFwc	iJ	**2689**	2725	FFwc
fe_sphere	iJ	**1730**	1745	FFwc	iJ	**2565**	2575	FFwc	JE	**3663**	3738	FFwc
cti	JE	**2915**	3082	FFwc	iJ	**4460**	4515	FFwc	iJ	**6125**	6365	FFwc
memplus	JE	**13742**	14590	FFwc	iJ	**14830**	15789	FFwc	JE	**17446**	17562	FFwc
cs4	JE	**2206**	2538	FF_2, FFraf	iJ	**3110**	3452	FF_2	iJ	**4223**	4711	FF_2
bcsstk30	JE	**72007**	75364	FFwc	iJ	120468	**120429**	FFwc	iJ	**180470**	183050	FFwc
bcsstk31	iJ	**25491**	25945	FFwc	iJ	**39440**	41038	FFwc	iJ	63626	**62621**	FFwc
bcsstk32	JE	**39682**	41842	FFwc	iJ	**65389**	67307	FFwc	JE	102820	**102694**	FFwc
t60k	N/A	**878**	973	FFwc	iJ	**1450**	1613	FF_2, FFraf, FFwc	iJ	**2277**	2484	FF_2, FFraf, FFwc
wing	JE	**4198**	4666	FF_2, FFraf, FFwc	iJ	**6074**	6700	FF_2, FFraf, FFwc	iJ	**8161**	9167	FFwc
brack2	JE	**12324**	12655	FF_2, FFraf, FFwc	iJ	**18421**	19146	FFwc	iJ	**27881**	28382	FFwc
finan512	Ch2.0	**1296**	1377	FF_2, FFraf, FFwc	Ch2.0	*2592*	*2592*	FF_2, FFraf, FFwc	pM4.0	11192	**10842**	FF_2, FFraf, FFwc
fe_tooth	iJ	**18435**	19346	FF_2, FFraf, FFwc	iJ	**26642**	27869	FFwc	iJ	**36487**	38385	FFwc
fe_rotor	iJ	**20773**	22961	FFwc	iJ	**33686**	34417	FFwc	iJ	50912	**50593**	FFwc
598a	iJ	**27344**	28021	FFwc	iJ	**41124**	42479	FFwc	iJ	**59152**	61737	FFwc
fe_ocean	iJ	**8507**	9028	FFwc	iJ	**13767**	14613	FF_2, FFraf, FFwc	iJ	**21854**	22777	FFwc
144	N/A	**39856**	42484	FFwc	iJ	**58659**	61505	FFwc	N/A	**81145**	85828	FFwc
wave	iJ	**46903**	48106	FF_2, FFraf, FFwc	iJ	**65926**	68348	FFwc	iJ	**91304**	92913	FFwc
m14b	N/A	**44515**	47702	FFwc	N/A	**68468**	73548	FFwc	N/A	**101980**	106389	FFwc
$< coupe\_min$			4	13%			3	10%			7	23%
$\leq coupe\_min$			4	13%			5	16%			7	23%

TAB. 7.11 – Meilleures partitions trouvées par l'algorithme de fusion-fission comparées aux résultats de la « Graph Partitioning Archive », pour une balance de partitionnement $balance = 1,01$

$bal = 1,03$	$k = 2$				$k = 4$				$k = 8$			
Graphe	Algo.	Coupe	Coupe	FF	Algo.	Coupe	Coupe	FF	Algo.	Coupe	Coupe	FF
add20	GCSVD	576	**571**	FFraf	JE	1188	**1181**	FFwc	iJ	**1720**	1721	FFwc
data	AMG	*185*	*185*	FF_2, FFraf, FFwc	iJ	383	**375**	FFwc	iJ	708	**658**	FFwc
3elt	JE	*87*	*87*	FF*	JE	*199*	*199*	FF*-FFraf	JE	**336**	339	FFwc
uk	JE	*18*	*18*	FF_1, kFF*	JE	**41**	42	kFF*	JE	**82**	94	kFF
add32	J2.2	*10*	*10*	FF*	JE	*33*	*33*	FF*	JE	69	**66**	FF_1
bcsstk33	AMG	*10064*	*10064*	FF*-FFraf-kFF*	iJ	21423	**20886**	FFwc	iJ	34322	**34200**	FFwc
whitaker3	JE	*126*	*126*	FF*	JE	**380**	381	FF_1, FFwc	JE	**658**	661	FFwc
crack	AMG	*182*	*182*	FF*-kFF*	JE	**360**	361	FFwc	JE	**676**	685	FFwc
wing_nodal	SDP	**1680**	1692	FFwc	JE	**3566**	3606	FFwc	JE	**5401**	5604	FFwc
fe_4elt2	MRSB	*130*	*130*	FF*	JE	349	**346**	FFwc	JE	**603**	617	FFwc
vibrobox	JE	**10310**	11184	FFwc	JE	19245	**19122**	FFwc	JE	24874	**24701**	FFwc
bcsstk29	GrPart	*2818*	*2818*	FF*-kFFraf	J2.2	8541	**8194**	FFwc	iJ	16505	**14659**	FFwc
4elt	JE	*137*	*137*	FF*-kFF*	N/A	**319**	320	FFwc	N/A	**527**	533	FFwc
fe_sphere	JE	*384*	*384*	FF_1, FF_2, FFwc	JE	**766**	768	FFwc	JE	*1152*	*1152*	FFwc
cti	JE	*318*	*318*	FF*	JE	*917*	*917*	FFwc	JE	**1716**	1800	FFwc
memplus	JE	**5407**	5479	FFwc	JE	**9664**	9676	FFwc	iJ	**11939**	12439	FF_1
cs4	AMG	**362**	404	FF_1	JE	**936**	1038	FF_1	N/A	**1472**	1682	FF_1
bcsstk30	JE	*6251*	*6251*	FF_1, FF_2, FFwc	JE	16617	**16588**	FF_1	JE	**34559**	35187	FFwc
bcsstk31	MLSAT	**2676**	2686	FF_1	pM4.0	7879	**7440**	FF_1	pM4.0	14426	**13906**	FFwc
bcsstk32	JE	**4667**	4739	FFraf	JE	9728	**9533**	FF_1	JE	**21307**	23537	FF_1
t60k	AMG	**71**	73	FF_2, FFraf	iJ	**211**	216	FF_1	N/A	**467**	496	FF_1
wing	MQI	**774**	870	FF_1, FF_2, FFraf	JE	**1636**	1838	FF_1	JE	**2551**	3001	FF_1
brack2	JE	*684*	*684*	FF_1	JE	**2864**	2899	FF_1	JE	**7080**	7492	FFwc
finan512	Ch2.0	*162*	*162*	FF*	Ch2.0	*324*	*324*	FF*	Ch2.0	*648*	*648*	FF*-FFraf
fe_tooth	SDP	**3792**	3858	FFwc	iJ	7435	**7176**	FFwc	iJ	12850	**12725**	FF_1
fe_rotor	SDP	**1965**	1966	FF_1	Ch2.0	8097	**7636**	FF_1	iJ	14028	**13588**	FF_1
598a	MQI	**2367**	2403	FF_1	N/A	**7978**	8220	FF_1	N/A	**16031**	17006	FF_1
fe_ocean	GrPart	**311**	312	FF_1, FFwc	N/A	**1704**	1836	FFraf	N/A	**4019**	4493	FFwc
144	MQI	**6438**	6655	FFwc	N/A	**15250**	16046	FF_1	N/A	**25611**	27872	FF_1
wave	SDP	**8616**	8848	FFwc	iJ	18058	**17599**	FF_1	iJ	**30895**	31311	FF_1
m14b	MQI	**3823**	3880	FFwc	JKM	13844	**13317**	FF_1	iJ	**27711**	27815	FF_1
$< coupe\_min$			1	3%			13	42%			8	26%
$\leq coupe\_min$			16	52%			17	55%			10	32%

TAB. 7.12 – Meilleures partitions trouvées par l'algorithme de fusion-fission comparées aux résultats de la « Graph Partitioning Archive », pour une balance de partitionnement $balance = 1,03$

$bal = 1,03$	$k = 16$				$k = 32$				$k = 64$			
Graphe	Algo.	Coupe	Coupe	FF	Algo.	Coupe	Coupe	FF	Algo.	Coupe	Coupe	FF
add20	iJ	2216	**2107**	FFwc	JE	2758	**2570**	FFwc	JE	3266	**3093**	FFwc
data	Ch2.0	1195	**1146**	FFwc	iJ	1970	**1889**	FFwc	iJ	**2911**	2918	FFwc
3elt	JE	**566**	575	FFwc	N/A	**958**	987	FFwc	N/A	**1552**	1596	FFwc
uk	N/A	**154**	171	FFwc	N/A	**265**	284	FFwc	iJ	*455*	*455*	FFraf
add32	JE	**117**	128	FF_2, FFraf, FFwc	JE	*212*	*212*	FFwc	pM4.0	624	**502**	kFF
bcsstk33	iJ	**55546**	56366	FFwc	iJ	**79121**	80112	FFwc	iJ	**109348**	111219	FFwc
whitaker3	JE	**1092**	1120	FFwc	JE	**1686**	1724	FFwc	JE	**2535**	2602	FFwc
crack	JE	**1082**	1130	FFwc	N/A	**1679**	1745	FFwc	JE	**2592**	2650	FFwc
wing_nodal	JE	**8316**	8652	FFwc	JE	**12024**	12426	FFwc	iJ	**16221**	16700	FFwc
fe_4elt2	JE	**1007**	1032	FFwc	JE	**1651**	1680	FFwc	iJ	**2537**	2605	FFwc
vibrobox	iJ	**32002**	34793	FFwc	iJ	**42582**	43190	kFF	JE	50849	**50105**	FFwc
bcsstk29	iJ	24706	**24147**	FFwc	iJ	**37810**	38031	FFwc	iJ	**58452**	59119	FFwc
4elt	JE	**916**	947	FFwc	JE	**1537**	1610	FFwc	JE	**2581**	2682	FFwc
fe_sphere	N/A	**1706**	1716	FFwc	JE	**2477**	2507	FFwc	JE	**3547**	3644	FFwc
cti	JE	**2778**	3018	FFwc	iJ	**4236**	4403	FFwc	iJ	**5992**	6245	FFwc
memplus	JE	**13559**	14507	FFwc	iJ	**14548**	15702	FFwc	JE	17409	**17361**	FFwc
cs4	N/A	**2126**	2517	kFF*	iJ	**3110**	3425	kFF*	iJ	**4196**	4583	kFF
bcsstk30	JE	**70768**	74833	FFwc	JE	**117232**	118805	FFwc	JE	**178578**	181212	FFwc
bcsstk31	iJ	**25160**	25542	FFwc	iJ	**38572**	40686	FFwc	iJ	**61066**	62254	FFwc
bcsstk32	JE	**38929**	40339	FFwc	iJ	**64302**	65805	FFwc	iJ	**96168**	101739	FFwc
t60k	N/A	**852**	973	FFwc	iJ	**1420**	1613	kFF*, FFwc	iJ	**2258**	2453	kFF*
wing	JE	**4015**	4666	FF_2, FFraf, FFwc	JE	**6028**	6700	kFF*, FFwc	iJ	**8161**	8932	kFF*
brack2	JE	**11958**	12655	FF_2, FFraf, FFwc	JE	**17952**	19052	FFwc	iJ	**27345**	28131	FFwc
finan512	Ch2.0	**1296**	1377	FF_2, FFraf, FFwc	Ch2.0	*2592*	*2592*	FF*	N/A	**10821**	10842	FF_2, FFraf, FFwc
fe_tooth	iJ	**18435**	19346	FF_2, FFraf, FFwc	iJ	**26642**	27760	FFwc	iJ	**36030**	38238	FFwc
fe_rotor	iJ	**20773**	22292	FFwc	iJ	**33686**	34279	FFwc	iJ	**47972**	50148	FFwc
598a	N/A	**26257**	28021	FFwc	N/A	**40792**	42412	FFwc	iJ	**58454**	61628	FFwc
fe_ocean	N/A	**7838**	8949	FFwc	N/A	**12746**	14569	FFwc	iJ	**21854**	22544	FFwc
144	N/A	**38478**	42265	FFwc	iJ	**58659**	61290	FFwc	N/A	**80894**	85679	FFwc
wave	iJ	**46487**	48106	FF_2, FFraf, FFwc	iJ	**64953**	68294	FFwc	iJ	**88383**	92755	FFwc
m14b	iJ	**44174**	47606	FFwc	N/A	**68468**	73528	FFwc	N/A	**101980**	106167	FFwc
$< coupe\_min$			3	10%			2	6%			4	13%
$\leq coupe\_min$			3	10%			4	13%			5	16%

TAB. 7.13 – Meilleures partitions trouvées par l'algorithme de fusion-fission comparées aux résultats de la « Graph Partitioning Archive », pour une balance de partitionnement $balance = 1,03$

$bal = 1,05$	$k=2$				$k=4$				$k=8$			
Graphe	Algo.	Coupe	Coupe	FF	Algo.	Coupe	Coupe	FF	Algo.	Coupe	Coupe	FF
add20	GCSVD	**550**	561	FFraf	iJ	1184	**1163**	FFwc	iJ	**1705**	1709	FFwc
data	SDP	*181*	*181*	kFFraf	iJ	378	**363**	FFwc	iJ	702	**642**	FFwc
3elt	JE	*87*	*87*	FF*	JE	199	**198**	FFwc	iJ	**334**	336	FFwc
uk	JE	*18*	*18*	FF*-FFwc	JE	*41*	*41*	FF_1, kFF	JE	**82**	93	FF_1
add32	J2.2	*10*	*10*	FF*	JE	*33*	*33*	FF*	JE	69	**65**	FF_1
bcsstk33	iJ	*9914*	*9914*	FF_1, FF_2, FFwc	iJ	21229	**20263**	FFwc	iJ	34117	**34103**	FFwc
whitaker3	JE	*126*	*126*	FF*	JE	*380*	*380*	FFwc	JE	*658*	*658*	FFwc
crack	AMG	*182*	*182*	FF*-kFF*	JE	**360**	361	FFwc	JE	**676**	679	FFwc
wing_nodal	SDP	**1668**	1681	FF_2	JE	**3566**	3606	FFwc	MLSAT	**5387**	5574	FFwc
fe_4elt2	MRSB	*130*	*130*	FF*	JE	349	**338**	FFraf	iJ	**597**	604	FFwc
vibrobox	JE	**10310**	11184	FFwc	JE	19245	**19001**	FFwc	iJ	**24158**	24470	FFwc
bcsstk29	GrPart	*2818*	*2818*	FF*-kFFraf	MLSAT	**8088**	8148	FF_1	MLSAT	15314	**14584**	FFwc
4elt	JE	*137*	*137*	FF*-kFF*	N/A	**319**	320	FFwc	N/A	**527**	530	FFwc
fe_sphere	JE	*384*	*384*	FF_1, FFwc	JE	*766*	*766*	FFwc	JE	*1152*	*1152*	FFwc
cti	JE	*318*	*318*	FF*	JE	917	**889**	FFwc	JE	**1716**	1732	FFwc
memplus	iJ	5353	**5339**	FFwc	MLSAT	**9427**	9453	FF_1	iJ	**11939**	12316	FF_1
cs4	AMG	**356**	383	FF_1	JE	**936**	1038	FF_1	N/A	**1472**	1682	FF_1
bcsstk30	JE	*6251*	*6251*	FF_1, FF_2, FFwc	JE	16617	**16433**	kFFraf	JE	**34559**	34806	FF_1
bcsstk31	MLSAT	**2676**	2686	FF_1	pM4.0	7879	**7399**	FF_1	iJ	**13561**	13906	FFwc
bcsstk32	JE	**4667**	4695	FF_1	JE	9728	**9052**	FF_1	JE	**21307**	23202	FF_1
t60k	SDP	**65**	67	FF_1	iJ	*211*	*211*	FF_1	N/A	**467**	496	FF_1
wing	MQI	**770**	870	FF_1, FF_2, FFraf	JE	**1636**	1838	FF_1	JE	**2551**	3001	FF_1
brack2	SDP	*660*	*660*	FF_1	MLSAT	2808	**2755**	FF_1	JE	**7080**	7492	FFwc
finan512	Ch2.0	*162*	*162*	FF*	Ch2.0	*324*	*324*	FF*	Ch2.0	*648*	*648*	FF*-FFraf
fe_tooth	SDP	**3773**	3858	FFwc	MLSAT	**7152**	7176	FFwc	iJ	**12646**	12725	FF_1
fe_rotor	SDP	**1957**	1966	FF_1	Ch2.0	8097	**7520**	FF_1	iJ	**13184**	13550	FFwc
598a	MQI	**2336**	2366	FF_1	N/A	**7978**	8179	FF_2, FFraf	N/A	**16031**	16942	FF_1
fe_ocean	GrPart	**311**	312	FF_1, FFwc	N/A	**1704**	1822	FF_1	N/A	**4019**	4433	FF_1
144	SDP	**6362**	6655	FFwc	N/A	**15250**	16046	FF_1	N/A	**25611**	27872	FF_1
wave	MQI	**8563**	8777	FF_1	iJ	18058	**17320**	FF_1	MLSAT	**30583**	31207	FF_1
m14b	MQI	**3802**	3878	FF_1	JKM	13844	**13317**	FF_1	iJ	**27711**	27815	FF_1
$< coupe\_min$			1	3%			14	45%			4	13%
$\leq coupe\_min$			16	52%			20	65%			7	23%

TAB. 7.14 – Meilleures partitions trouvées par l'algorithme de fusion-fission comparées aux résultats de la « Graph Partitioning Archive », pour une balance de partitionnement $balance = 1,05$

$bal = 1,05$	$k = 16$				$k = 32$				$k = 64$			
Graphe	Algo.	Coupe	Coupe	FF	Algo.	Coupe	Coupe	FF	Algo.	Coupe	Coupe	FF
add20	MLSAT	2186	**2097**	FFwc	JE	2758	**2539**	FFwc	JE	3266	**3093**	FFwc
data	Ch2.0	1195	**1133**	FFwc	iJ	1922	**1862**	FFwc	iJ	**2911**	2918	FFwc
3elt	JE	**566**	571	FFwc	N/A	**958**	968	FFwc	N/A	**1552**	1573	FFwc
uk	N/A	**154**	171	FF_2, FFwc	N/A	**265**	283	FF_2	iJ	**436**	452	FF_2, FFraf
add32	JE	**117**	128	FF_2, FFraf, FFwc	JE	*212*	*212*	FFwc	pM4.0	624	**493**	FFwc
bcsstk33	iJ	**55546**	56195	FFwc	iJ	**78821**	79579	FFwc	iJ	**108249**	110623	FFwc
whitaker3	JE	**1092**	1120	FFwc	JE	**1686**	1719	FFwc	JE	**2535**	2587	FFwc
crack	JE	**1082**	1123	FFwc	N/A	**1679**	1734	FFwc	iJ	**2590**	2612	FFwc
wing_nodal	JE	**8316**	8625	FFwc	JE	**12024**	12382	FFwc	iJ	**16102**	16643	FFwc
fe_4elt2	JE	**1007**	1021	FFwc	JE	**1651**	1679	FFwc	iJ	**2516**	2590	FFwc
vibrobox	iJ	**31695**	34515	FFwc	iJ	**41176**	43019	kFF	iJ	50757	**49574**	FFwc
bcsstk29	iJ	24706	**23620**	FFwc	iJ	**36731**	37876	FFwc	iJ	**58108**	58913	FFwc
4elt	JE	**916**	943	FFwc	JE	**1537**	1591	FFwc	JE	**2581**	2654	FFwc
fe_sphere	iJ	**1692**	1702	FFwc	JE	**2477**	2480	FFwc	JE	**3547**	3592	FFwc
cti	JE	**2778**	2980	FFwc	iJ	**4236**	4371	FFwc	iJ	**5907**	6200	FFwc
memplus	iJ	**13279**	14428	FFwc	iJ	**14384**	15659	FFwc	JE	17409	**17251**	FFwc
cs4	N/A	**2126**	2517	kFF*	iJ	**3080**	3425	kFF*	iJ	**4196**	4563	kFF
bcsstk30	JE	**70768**	74758	FFwc	JE	**117232**	118078	FFwc	iJ	**177379**	180503	FFwc
bcsstk31	iJ	**24179**	25430	FFwc	iJ	**38572**	40634	FFwc	iJ	**60446**	62130	FFwc
bcsstk32	iJ	**38320**	40317	FFwc	iJ	**62961**	65705	FFwc	iJ	**96168**	101531	FFwc
t60k	N/A	**852**	954	kFF*	iJ	**1420**	1589	kFF*	iJ	**2221**	2453	kFF*
wing	JE	**4015**	4666	FF_2, FFraf, FFwc	iJ	**6010**	6632	kFF*	iJ	**8161**	8932	kFF*
brack2	JE	**11958**	12655	FF_2, FFraf, FFwc	JE	**17952**	19052	FFwc	iJ	**26944**	28039	FFwc
finan512	Ch2.0	**1296**	1377	FF_2, FFraf, FFwc	Ch2.0	*2592*	*2592*	FF*	N/A	**10821**	10842	FF_2, FFraf, FFwc
fe_tooth	iJ	**18435**	19346	FF_2, FFraf, FFwc	MLSAT	**26016**	27749	FFwc	iJ	**36030**	38222	FFwc
fe_rotor	iJ	**20773**	22206	FFwc	iJ	**33686**	34115	FFwc	iJ	**47852**	50010	FFwc
598a	N/A	**26257**	28021	FFwc	iJ	**40179**	42412	FFwc	iJ	**58307**	61628	FFwc
fe_ocean	N/A	**7838**	8879	FFwc	N/A	**12746**	14510	FFwc	iJ	**21784**	22463	FFwc
144	N/A	**38478**	42196	FFwc	N/A	**58142**	61242	FFwc	N/A	**80445**	85655	FFwc
wave	MLSAT	**44625**	48106	FFwc	iJ	**63725**	68294	FFwc	iJ	**88383**	92755	FFwc
m14b	iJ	**44174**	47606	FFwc	N/A	**68468**	73528	FFwc	iJ	**101385**	106145	FFwc
$< coupe\_min$			3	10%			2	6%			4	13%
$\leq coupe\_min$			3	10%			4	13%			4	13%

TAB. 7.15 – Meilleures partitions trouvées par l'algorithme de fusion-fission comparées aux résultats de la « Graph Partitioning Archive », pour une balance de partitionnement $balance = 1,05$

bleau que celui des archives du partitionnement de graphe, c'est-à-dire nous avons trié les résultats en fonction de la balance de partitionnement, puis du nombre de parties de la partition. Chaque partition de ces archives est décrite avec son coût de coupe et l'outil qui a permis de la trouver (le tableau 7.6 permet de faire la correspondance entre l'abréviation désignant un de ces outils et son nom). Chacune d'elles fait face au meilleur coût de coupe obtenu par l'une des implémentations de la fusion-fission, et à l'acronyme de celle-ci. Afin de condenser la présentation, nous avons utilisé les notations suivantes :

– FF* désigne toutes les implémentations de la fusion-fission ;
– kFF* désigne kFF et kFFraf ;
– FF*-kFF désigne toutes les implémentations sauf kFF et ainsi de suite.

Nous rappelons les différents temps d'exécution de ces implémentations :

– FF_1 : de 6 à 12 heures ;
– FFwc : 4 heures ;
– FF_2, FFraf, kFF et kFFraf : 2 heures.

L'algorithme de fusion-fission étant auto-suffisant, c'est le seul paramètre qui lui est nécessaire avec le graphe et le nombre de parties de la partition (l'algorithme cherche des partitions répondant aux 4 balances utilisées par ces archives).

Comme 5 des outils de partitionnement de ces archives sont des métaheuristiques appliquées au partitionnement de graphe contraint (*cf* tableau 7.6), la comparaison que nous faisons dans cette section se rapporte aussi à une comparaison entre la fusion-fission et plusieurs métaheuristiques. De plus, 4 de ces outils sont aussi des implémentations itératives de méthodes multi-niveaux, à l'image de celle décrite dans la section précédente (*supra* 7.3). Si certains des outils du tableau 7.6 ont des temps de calcul particulièrement rapides, comme les logiciels pMetis, Jostle ou Chaco (au plus quelques secondes), d'autres ont des temps de calcul plus longs que la fusion-fission, comme la méthode Jostle évolutionnaire (jusqu'à plusieurs semaines [WAL04]). Comme on peut s'y attendre, ce sont les méthodes dont le temps de calcul est le plus long qui permettent de trouver le plus de partitions de faible coût : 271 pour Jostle évolutionnaire (JE) ; 255 pour *iterated Jostle* (iJ) ; 36 pour Chaco (Ch2.0) ; 14 pour pMetis (pM4.0) ; et 4 pour Jostle (J2.2).

Les deux dernières lignes des tableaux 7.8 à 7.13 récapitulent : sur la première ligne, le nombre de fois où au moins une implémentation de la fusion-fission a permis de trouver une partition de coût de coupe strictement inférieur à celui des archives ; sur la seconde ligne, le nombre de fois où au moins une implémentation de la fusion-fission a permis de trouver une partition de coût de coupe inférieur ou égal à celui des archives. En faisant le total de ces occurrences, nous constatons que 136 partitions de coût de coupe inférieurs ont été trouvées, soit 18,3% du nombre total de partitions (744), et que 228 partitions de coût de coupe au moins égal ont été trouvées, soit 30,6% des partitions.

Le tableau 7.6 présente un récapitulatif des nombres de fois où une implémentation de la fusion-fission a trouvé une partition de qualité soit meilleure, soit au moins égale à celle des archives. À l'inverse des autres tableaux, ce récapitulatif est présenté en fonction du nombre de parties de la partition cherchée et non de la balance de partitionnement. Il montre que le nombre de parties qui réussit le plus à la fusion-fission est 4. Il nous permet aussi de remarquer que la fusion-fission trouve plus de bonnes solutions pour des partitions en 64 parties qu'en 16 et 32 parties, ce qui peut nous faire penser que l'adaptation de la séquence des nombres de parties au cas où k est grand fonctionne assez bien. De plus, si on revient sur la remarque que nous avions faite concernant cette adaptation un peu plus haut (*supra* 7.2) et qui préconisait d'utiliser celle-ci de manière moins restrictive, alors cela nous ouvre des pistes prometteuses vers une amélioration de l'algorithme de fusion-fission.

7.5 Conclusion

Ce chapitre a comparé de manière exhaustive les résultats trouvés par l'application de la méthode de fusion-fission au partitionnement de graphe contraint avec ceux trouvés par les outils et méthodes de la littérature. Pour celà, des bancs de tests classiques du partitionnement de graphe ont été utilisés.

Dans un premier temps, la robustesse de l'algorithme de fusion-fission a été testées ; elle s'est révélée plus élevée que celle des outils multi-niveaux. Dans un second temps ont été évalués les gains de performance de l'algorithme de fusion-fission comparativement aux résultats obtenus après une itération des outils multi-niveaux. Une troisième section a comparé les différentes améliorations apportées à l'algorithme de fusion-fission. Il est apparu que l'implémentation utilisant la méthode d'affinage de Walshaw-Cross était la plus performante. Enfin, la dernière section a présenté les meilleurs résultats obtenus par les différentes implémentations de la fusion-fission et les a comparés avec les meilleures partitions trouvées actuellement des archives sur le partitionnement de graphe.

Ce chapitre a montré les bons résultats obtenus par la méthode de fusion-fission sur le problème du partitionnement contraint, en les comparant aux autres méthodes de la littératures. Il a prouvé que si l'algorithme de fusion-fission était bien plus lent que les algorithmes multi-niveaux, il était au moins aussi rapide que les métaheuristiques hybrides utilisant une méthode multi-niveaux ; de plus, cet algorithme a permis de trouver un bon nombre de résultats meilleurs que ceux de ces méthodes hybrides. De nombreuses partitions dont le coût de coupe n'avait jamais été atteint ont été trouvées par cet algorithme. De nouvelles pistes de recherche concernant l'algorithme de fusion-fission sont aussi proposées dans ce chapitre. Celles-ci pourraient permettre d'améliorer les performances de cet algorithme dans le futur.

Conclusions et perspectives

Plusieurs méthodes et outils de partitionnement ont été présentés dans cette thèse, puis appliqués au découpage de l'espace aérien européen, à des bancs de tests classiques ainsi qu'à la segmentation d'images et à la classification de textes.

Méthodes

Avant de s'intéresser aux méthodes de partitionnement de graphe, et dans la perspective des applications que nous avons faites, nous avons proposé un classement regroupant en deux catégories les différents problèmes de partitionnement de graphe :

– le problème du **partitionnement de graphe contraint** a pour objectif de trouver une partition d'un graphe ayant un coût de coupe minimal sous la contrainte : cette partition doit respecter une balance de partitionnement ainsi qu'un nombre de parties déterminé à l'avance ;
– le problème du **partitionnement de graphe non contraint** a pour objectif de minimiser une fonction de coût comme la coupe normalisée ou le ratio de coupe et pour contrainte de trouver une partition formée d'un nombre de parties déterminé à l'avance.

Alors que le problème de partitionnement des bancs de tests classiques fait partie de la catégorie du partitionnement de graphe contraint, le problème du découpage de l'espace aérien est celui du partitionnement de graphe non contraint. Les deux autres problèmes que nous avons abordés, à savoir la segmentation d'image et la classification de documents, sont aussi des problèmes de partitionnement de graphe non contraint.

Cette thèse a étudié de nombreuses méthodes de partitionnement de graphe, des plus classiques comme l'expansion de région, la méthode spectrale ou l'approche multi-niveaux, jusqu'à une nouvelle méthode, la fusion-fission, en passant par des métaheuristiques :

– les méthodes d'**expansion de région** présentées dans cette thèse sont mises en œuvre soit par les méthodes multi-niveaux, comme c'est le cas des algorithmes *graph growing* (GGP) et *greedy graph growing* (GGGP), soit par la fusion-fission, comme c'est le cas pour notre algorithme basé sur la méthode de percolation ;
– nous avons décrit la méthode **spectrale**, qui est basée sur le calcul des vecteurs propres de la matrice d'adjacence du graphe. Elle a été très utilisée dans le cas du partitionnement de graphe contraint, mais est maintenant remplacée par la méthode multi-niveaux. Cependant, dans le cas du partitionnement non contraint, elle sert encore souvent ;
– plusieurs méthodes d'affinage basées sur l'**algorithme de Kernighan-Lin** sont décrites, ainsi que la notion de gain associée. Après avoir analysé cet algorithme, nous avons présenté l'implémentation de Fiduccia-Mattheyses qui lui permet d'être plus rapide. Cet algorithme a été adapté de la bissection au partitionnement en plusieurs parties grâce aux algorithmes *global Kernighan-Lin refinement* (GKLR) et de Walshaw-Cross. Nous avons aussi utilisé cette notion de gain pour

créer un nouvel algorithme de répartition de charge ;
- la méthode **multi-niveaux**, qui consiste à réduire la taille du graphe avant de le partitionner, a été décortiquée afin d'en connaître tous ses mécanismes. Elle est actuellement la méthode la plus rapide et la plus performante pour le partitionnement contraint des graphes de plus d'un millier de sommets, mais son adaptation au cas non contraint est encore marginale. Parmi les logiciels de référence, tous l'utilisent : Chaco, Metis, Scotch, Jostle ;
- la première méthode que nous avons utilisée pour résoudre le problème du découpage de l'espace aérien a été un **recuit simulé**. Cette métaheuristique très classique a été décrite puis appliquée deux fois, de manière différente, à ce problème ;
- la métaheuristique des **colonies de fourmis** a elle aussi été adaptée au problème du découpage de l'espace aérien. Cette adaptation met en concurrence plusieurs colonies pour le partage du graphe ;
- les **algorithmes évolutionnaires**, très étudiés dans notre laboratoire, ont été utilisés pour résoudre le problème du partitionnement de graphe non contraint, mais aussi contraint ;
- enfin, nous avons présenté une nouvelle méthode basée sur une analogie avec le mécanisme de fusion et de fission d'atomes en physique nucléaire ; nous l'avons nommée **fusion-fission** en référence à ce processus. Cette méthode, qui est la principale innovation de la thèse, a été appliquée avec succès au partitionnement de graphe contraint et non contraint.

Applications

Après avoir présenté ces différentes méthodes de partitionnement de graphe, nous les avons utilisées pour résoudre le problème du découpage de l'espace aérien européen en centres de contrôle et en zones de qualification. Ce découpage est basé sur les flux de trafic actuel et le découpage existant en secteurs de contrôle. Les données de trafic et de découpage nous ont été fournies par Eurocontrol ; leur traitement, nécessairement long, a été rendu difficile par les problèmes de cohérence dans les données. La comparaison des différentes méthodes de partitionnement non contraint sur ce problème a montré que la fusion-fission permettait de trouver les meilleures partitions du ciel au regard de la fonction objectif et des contraintes utilisées. Cette fonction objectif, le ratio de coupe, a été choisie afin de minimiser la charge de coordination des contrôleurs et donc d'augmenter la sûreté du contrôle. Les contraintes que nous avons utilisées proviennent des recommandations d'Eurocontrol sur la taille des centres de contrôle. Cette fonction objectif et ces contraintes sont-elles les mieux adaptées au niveau opérationnel pour optimiser le découpage de l'espace aérien ? Des recherches ultérieures en fourniront peut-être d'autres, convenant d'avantage à cet important problème concret, que nous n'avons pas la prétention d'avoir définitivement résolu. La méthode de fusion-fission peut facilement être adaptée à une autre fonction objectif, ainsi qu'à d'autres contraintes, c'est là un de ses avantages.

Pour illustrer les résultats des méthodes de partitionnement de graphe, nous avons proposé trois découpages de l'espace aérien en centres de contrôle et en zones de qualification :
- le premier découpage est celui de la France. Visuellement, les résultats trouvés ont une certaine similarité avec le découpage existant, ce qui conforte celui-ci ;
- comme notre sujet se place dans le cadre de la création de blocs fonctionnels d'espace aérien (FAB) prévue par la Commission européenne, nous avons aussi étudié un redécoupage du FAB d'Europe centrale ;
- enfin, pour avoir une vue d'ensemble des régions de forte densité de trafic en Europe, nous avons aussi présenté un redécoupage de la *core area*.

Afin de diversifier les applications de la méthode de fusion-fission, nous l'avons appliquée à la segmentation d'images et à la classification de documents.

Sur ce premier problème, la fusion-fission permet d'isoler assez nettement un objet sur une image simple. Le résultat est visuellement bon en comparaison de la segmentation obtenue avec d'autres méthodes utilisant la même fonction de coût. Cependant, pour des images plus complexes, l'approche que nous avons utilisée montre ses limites. Une adaptation de la fusion-fission à une autre fonction de coût serait certainement à envisager pour obtenir de meilleurs résultats.

L'application de la fusion-fission à la classification de documents a été faite en se basant sur un modèle de classification existant. Les résultats obtenus par la fusion-fission montrent les limites de ce modèle de classification. En effet, en appliquant la fusion-fission à une base de données de textes de disciplines différentes, celle-ci trouve un classement de meilleure qualité que celui d'origine, fait par discipline. Là encore, c'est la fonction de coût qui semble mal adaptée. Il serait donc intéressant d'étudier et de proposer pour ce modèle d'autres fonctions de coût.

Enfin, dans le but d'évaluer la qualité de la méthode de fusion-fission, nous l'avons comparée aux résultats des bancs de tests classiques du partitionnement de graphe contraint. Ceux-ci proviennent des *archives du partitionnement de graphe*[2] maintenues par Chris Walshaw. Cette base de données de 34 graphes contient les partitions de coût de coupe les plus bas trouvées par 20 outils de partitionnement. Dans presque un cinquième des tests, la fusion-fission a trouvé une partition de coût de coupe strictement inférieur, et dans près du tiers des tests, une partition de coût de coupe au plus égale. Cependant, ces résultats ont aussi permis de repérer quelques défauts de notre algorithme. Le premier concerne son temps d'exécution qui est très long, en effet, ces résultats ont été obtenus après plusieurs heures de calcul. Le second concerne la recherche de partitions de plus de 8 parties. Nous avons aussi pu observer que la fusion-fission était plus robuste que les méthodes multi-niveaux et permettait de trouver des partitions de meilleure qualité que celles-ci, même lorsque les résultats de ces dernières étaient sélectionnés après plusieurs milliers d'exécutions.

Perspectives

Comme nous l'avons déjà souligné, la modélisation du problème de découpage aérien que nous avons réalisée n'est pas définitive. En effet, d'autres paramètres comme la complexité du trafic, un redécoupage des secteurs ou une nouvelle modélisation des routes aériennes pourraient être pris en compte dans une étude plus vaste et plus complexe que cette thèse. Une telle étude, portant sur plus de paramètres, et mise en œuvre par une équipe d'experts du domaine, serait la continuité logique de cette thèse, et pourrait s'appuyer sur les méthodes et outils qui y sont présentés, principalement la méthode de fusion-fission. Ainsi, les perspectives d'utilisation opérationnelle de ce travail ne se situent pas à court terme, mais à moyen et long termes (au moins quelques années).

Les perspectives d'évolution de la méthode de fusion-fission sont nombreuses :

- l'adaptation de la fusion-fission à d'autres problèmes d'optimisation combinatoire est pour nous une priorité. Si cette adaptation se révèle possible et intéressante, alors la fusion-fission pourra être généralisée à la manière d'une métaheuristique. L'adaptation de la méthode multi-niveaux à d'autres problèmes d'optimisation combinatoire nous donne quelques pistes sur une future adaptation et nous permet d'espérer que celle-ci est possible ;
- d'autres évolutions de la méthode et de l'algorithme de fusion-fission sont possibles dans le domaine du partitionnement de graphe :
  - la rapidité de l'algorithme mériterait d'être améliorée ;

[2] *The Graph Partitioning Archive* est disponible à l'adresse : staffweb.cms.gre.ac.uk/ wc06/partition/

  - il faudrait résoudre le problème de la dégradation des performances lorsque le nombre de parties augmente ;
  - d'autres outils de partitionnement performants comme le logiciel Scotch pourraient être utilisés par la fusion-fission ;
  - cette méthode pourrait être adaptée au calcul parallèle.
- il serait possible d'appliquer la fusion-fission à des problèmes proches mais différents du partitionnement de graphe, comme nous avons commencé à le faire pour la segmentation d'image et la classification de documents. Des travaux récents [DHI07] montrent la possible adaptation de méthodes de partitionnement de graphe aux problèmes de partitionnement de données (*data clustering* en anglais). En s'inspirant de ces travaux, de nouveaux champs d'application de la fusion-fission sont envisageables.

Annexes

Annexe A

Les principaux outils et bancs de tests pour le partitionnement de graphe

Cette annexe présente des outils de partitionnement de référence, développés par différents laboratoires à travers le monde. Cependant, cette énumération est loin d'être exhaustive. En dehors d'avoir été conçu pour résoudre le problème du partitionnement de graphe, la caractéristique commune de ces outils est qu'ils inclus tous un algorithme multi-niveaux. La description de ces outils est suivie par la présentation de bancs de tests pour le partitionnement de graphe.

A.1 Les outils pour le partitionnement contraint

Les calculs d'ingénierie et principalement les opérations matricielles demandent toujours plus de puissance et de rapidité. Pour répondre à ces besoins, des machines parallèles ou des *clusters* d'ordinateurs sont utilisés. Il faut alors distribuer la charge de calcul entre ces machines. Bon nombre d'outils de partitionnement classiques ont été créés pour calculer la partition nécessaire à ces distributions. Le problème de partitionnement sous-jacent est un problème de partitionnement contraint. La plupart des outils présentés dans cette section servent dans ce contexte.

Dans cette section ont été sélectionnés les outils remplissant les quatre conditions suivantes :

- résoudre le problème du partitionnement contraint ;
- une grande notoriété ;
- la gratuité ;
- de bonnes performances.

Cinq bibliothèques ont ainsi été sélectionnées : Chaco, Metis, Scotch, Jostle, Party.

A.1.1 Chaco

La bibliothèque Chaco [HEN95a] a été écrite par Bruce Hendrickson et Robert Leland des *Sandia National Laboratories* aux États-Unis. Elle est disponible à l'adresse :
www.cs.sandia.gov/ bahendr/chaco.html .

C'est la plus ancienne des bibliothèques présentées dans cette sous-section. Elle comporte un large choix de techniques de résolution pour le partitionnement contraint :

- un algorithme spectral (*supra* 2.3) utilisant deux méthodes de résolution [POT90, HEN95b], l'algorithme de Lanczos ou un algorithme associant la méthode itérative du quotient de Rayleigh et la méthode multi-niveaux.

– un algorithme de Kernighan-Lin généralisé pour le k-partitionnement direct d'un graphe pondéré et utilisant l'implémentation de Fiduccia-Mattheyses. Cet algorithme est décrit dans [HEN95c] ;
– l'algorithme multi-niveaux de [HEN95c] ;
– un algorithme utilisant une méthode inertielle basée sur des informations géométriques venant s'ajouter à la description du graphe [SIM91, WIL91].

Le code source de cette bibliothèque est disponible. Entièrement écrite en langage C, elle possède une interface facile à utiliser. Les méthodes implémentées dans Chaco cherchent toutes à résoudre le partitionnement de graphe contraint. Cette application est hautement paramétrable mais souffre d'un inconvénient majeur : le cardinal de la partition cherchée doit être une puissance de deux. L'un des paramètres les plus importants se nomme KL_IMBALANCE et permet de modifier la balance de partitionnement de la partition cherchée.

A.1.2 Metis

La bibliothèque Metis [KAR98b] a été créé par George Karypis et Vipin Kumar de l'Université du Minnesota aux États-Unis d'Amérique. Toutes les informations sur Metis sont disponibles à l'adresse : glaros.dtc.umn.edu/gkhome/views/metis .

Metis est l'une des bibliothèques les plus utilisées pour le partitionnement de graphe. Dans cette bibliothèque, deux logiciels vont nous intéresser : pMetis et kMetis. Le premier logiciel, pMetis, utilise un algorithme multi-niveaux récursif de bissection décrit dans [KAR98a]. Le second logiciel, kMetis, est implémenté avec un algorithme de k-partitionnement direct présenté dans [KAR98d]. Le code de cette bibliothèque est entièrement disponible, bien que non libre. Elle est écrite en langage C et possède une interface très accessible. Malheureusement, ces logiciels est beaucoup moins paramétrables que le logiciel Chaco. À titre d'exemple, la balance de partitionnement n'est pas paramétrable. Ainsi, pMetis cherche des partitions de balance $1,01$ et kMetis de balance $1,03$.

A.1.3 Scotch

La bibliothèque Scotch [PEL06] est maintenue par François Pellegrini de l'INRIAet de l'ENSEIRBen France. Cette bibliothèque , sa documentation et d'autres ressources autour de Scotch sont disponibles à l'adresse : www.labri.fr/perso/pelegrin/scotch/ .

Scotch comporte un ensemble abondant de programmes autour du partitionnement de graphe pour une utilisation en calcul numérique. Ainsi, Scotch fournit des outils d'affectation statique de tâche (*static mapping*), de partitionnement et de réorganisation matricielle (*matrix ordering*). Sa fonction de coût peut, en plus du coût de coupe, tenir compte de spécificités particulières du problème de partitionnement [PEL96, HEN97]. L'outil de partitionnement de Scotch utilise un algorithme multi-niveaux de bissection récursive avec une méthode d'affinage de type Kernighan-Lin décrite dans [PEL96]. Contrairement aux autres bibliothèques, qui utilisent comme format de graphe celui de Chaco, Scotch utilise son propre format. Cependant un programme de Scotch permet de passer facilement d'un format à l'autre. La bibliothèque codée en langage C dispose de très nombreuses interfaces de programmation.

L'un des grands avantages de l'outil de partitionnement de Scotch est son polymorphisme. En effet, dans celui-ci, la méthode de partitionnement peut être réglée précisément. De plus, il n'est pas restrictif sur la balance de partitionnement à prendre en compte.

A.1.4 Jostle

Le logiciel Jostle [WAL02a] a été écrit par Chris Walshaw de l'Université de Greenwich au Royaume-Uni. La bibliothèque Jostle n'est plus disponible ; seul l'exécutable l'est. Les sources de celui-ci ne sont pas disponibles non plus. Pour obtenir une copie du logiciel Jostle, il faut faire parvenir à M. Walshaw un exemplaire signé de la licence. La documentation relative à cet exécutable et à sa licence est disponible à l'adresse :
staffweb.cms.gre.ac.uk/ c.walshaw/jostle .

Jostle est un logiciel basé sur l'algorithme multi-niveaux décrit dans [WAL00a]. Les possibilités d'utilisation de Jostle sont réduites du fait qu'il ne possède pas d'interface de programmation. Cependant, il est assez paramétrable : le cardinal de la partition cherchée n'est pas réduit aux puissances de deux comme Chaco ; sa balance de partitionnement est très ajustable, allant de $1,00$ à $1,50$; et son algorithme d'affinage peut être utilisé indépendamment.

A.1.5 Party

La bibliothèque Party [PRE98] est due à Robert Preis de l'Université de Paderborn en Allemagne. La page internet de la bibliothèque Party est accessible à l'adresse :
wwwcs.uni-paderborn.de/fachbereich/AG/monien/RESEARCH/PART/party.html .

La bibliothèque Party fournit un logiciel de partitionnement assez atypique. En effet, celui-ci dispose nativement de huit méthodes de partitionnement, dont une optimale et deux d'expansion de région. De plus, Party peut utiliser toutes les méthodes disponibles dans Chaco. Party dispose de deux méthodes d'affinage :

– un algorithme type Kernighan-Lin utilisant l'implémentation de Fiduccia-Mattheyses ;
– un algorithme utilisant la méthode *Helpful-Sets* décrite dans [DIE95, SCH03, MON04].

Party permet aussi d'ajuster la balance de partitionnement.

A.2 Les outils pour le partitionnement non contraint

Les outils pour le partitionnement de graphe non contraint (*supra* 1.6) sont moins nombreux que dans le cas contraint. De plus, ce sont souvent des outils de segmentation d'image qui utilisent la notion de distance entre sommets, ce qui les rend inutilisables dans notre cas.

L'outil de segmentation d'image le plus connu est celui de Jianbo Shi et Jitendra Malik [SHI00]. Leur logiciel utilise une méthode spectrale décrite pour minimiser le coût de coupe normalisée.

A.2.1 Graclus

Le logiciel Graclus [DHI07, DHI04c, DHI04a] est une adaptation du logiciel pMetis au cas non contraint. Cette adaptation a été réalisée par Inderjit Dhillon, Yuqiang Guan et Brian Kullis de l'université du Texas. Le logiciel et les articles décrivant les méthodes utilisées par Graclus sont disponibles à l'adresse :
www.cs.utexas.edu/users/dml/Software/graclus.html .

L'implémentation de Graclus est basée pour le partitionnement, sur l'algorithme multi-niveaux de pMetis, et pour l'affinage, sur l'algorithme des centres mobiles (*supra* 2.1.3). Ce dernier algorithme est habituellement utilisé pour regrouper des individus qui peuvent être représentés sous forme de points de R^p muni d'une distance euclidienne. Cependant, Inderjit Dhillon, Yuqiang Guan et Brian Kulis proposent dans [DHI07] une notion de distance entre les sommets d'un graphe qui permet d'utiliser cette méthode.

A.3 Performance des différents algorithmes multi-niveaux

Le site internet de Chris Walshaw, intitulé *the graph partition archive*, abrite une importante base de données sur le partitionnement de graphe contraint. Il est consultable à l'adresse :
staffweb.cms.gre.ac.uk/ wc06/partition/ . Sa base de données comporte 34 graphes de différentes natures et venant de source multiples. En juin 2007, 20 outils ou algorithmes de partitionnement de graphe contraint y sont mis en compétition. Parmi ces outils, plusieurs ont déjà été présentés (*supra* A.1) : Chaco, Metis et Jostle. Les meilleures partitions trouvées par ces outils sont disponibles sur le site. Les différentes partitions cherchées ont pour cardinal $2, 4, 8, 16, 32$ ou 64, et comme balance de partitionnement $1,00, 1,01, 1,03$ ou $1,05$. Ainsi, ce ne sont pas moins de $34 * 6 * 4 = 816$ partitions qui sont consultables.

Ce site est plusieurs fois mentionné dans cette thèse car il représente, à notre connaissance, la plus grande base de partitions disponible actuellement. La richesse des informations qui y sont contenues permet de donner une première évaluation de la performance des algorithmes de partitionnement contraint. La section qui suit présente une version de ce site datant du 1erjuin 2007.

A.4 Archives du partitionnement de graphe de Chris Walshaw

Cette section présente la version du 1erjuin 2007 des archives du partitionnement de graphe de Chris Walshaw disponible à l'adresse : staffweb.cms.gre.ac.uk/ wc06/partition/ .

The Graph Partitioning Archive

Welcome to the University of Greenwich Graph Partitioning Archive. The archive consists of the best partitions found to date for a range of graphs and its aim is to provide a benchmark against which partitioning algorithms can be tested and as a resource for experimentation.

Partitions computed for the colouring test suite in [Wal04] are contained in a separate annexe.

The archive was initially set up as part of a research project into very high quality partitions and authors wishing to refer to the partitioning archive should cite the paper [SWC04].

Problem definition

Let $G = G(V, E)$ be an undirected graph of vertices V, with edges E. We assume that both vertices and edges can be weighted and that $|v|$ denotes the weight of a vertex v and similarly for edges and sets of vertices and edges. However, it is often the case that vertices and edges are given unit weights, $|v| = 1$ for all v in V and $|e| = 1$ for all e in E and indeed so far this is true for all of the example graphs. A partition of the graph is a mapping of V into P disjoint subdomains S_p such that the union of $S\_p = V$. The weight of a subdomain is just the sum of the weights of the vertices in the subdomain, $|S\_p| = sum|v|$ and the set of inter-subdomain or cut edges (i.e. edges cut by the partition) is denoted by E_c.

The usual objective of graph partitioning is to find a partition which evenly balances the vertex weight in each subdomain whilst minimising the total weight of cut edges or cut-weight, $|E\_c|$. To evenly balance the vertex weight, the optimal subdomain weight is given by $S\_opt := ceil(|V|/P)$ (where the ceiling function $ceil(x)$ returns the smallest integer greater than x). The graph partitioning problem can then be specified as : find a partition of G such that $|E\_c|$ is minimised subject to the constraint that $|S\_p| <= S\_opt$.

In fact it has been noted for some time that partition quality can often be improved if a certain amount of imbalance is allowed, [ST97]. The imbalance is defined as the maximum subdomain weight, $S' = \max |S\_p|$, divided by the optimal, S_opt. If we allow $x\%$ imbalance then the partitioning problem becomes : find a partition of G such that $|E\_c|$ is minimised subject to the constraint that $|S\_p| <= S\_opt.(100 + x)/100$.

The graph partitioning problem arises in many applications such as VLSI circuit design, data mining and parallel computing. In the latter case the subdomains are mapped to processors and the cut-weight then approximates the total communication volume. However, there is some discussion about whether this is the most appropriate metric for partitioning, e.g. [HK00], and indeed it is unlikely that any one metric is appropriate. Currently in the archive, the quality of partitions is determined by cut-weight, however, there is no reason why it should not be extended to incorporate partitions which minimise different cost functions.

Submissions

The archive welcomes submissions of either partition files, graph files or both. Partition files will be validated and added to the archive if they improve on a particular result. In this context improvement is :

- having a lower cut-weight than the exisiting result whilst not exceeding the maximum subdomain weight ; or
- having the same cut-weight as an exisiting result but a smaller maximum subdomain weight ; or
- being a new result.

File for submission should be made available on a web site or ftp site and a mail message indicating their presence sent to c.walshaw@gre.ac.uk. If this is not possible the files may be mailed directly. Submitted files should also include some indication of how the partition was derived which, if a public domain software package, may be subject to verification. The archive reserves the right not to accepted submitted files (either for reasons of disk space or for other unspecified reasons).

Archived partitions

The best partitions found to date are listed in 4 tables corresponding to 0% imbalance (perfect balance), 1% imbalance, 3% imbalance and 5% imbalance. For each graph, partitions into 2, 4, 8, 16, 32 and 64 subdomains are given and have the format C (S') [M] where C is the cut-weight, S' is the weight of the largest subdomain and M is the method/software that produced the partition. The cut-weight also provides a link to the partition file. The format of the partition file is a list of $|V|$ integers p, with $0 <= p < P$, and each denoting the subdomain to which the corresponding vertex has been assigned (e.g. if the integer on line 200 is 0 then vertex number 200 is in subdomain 0).

Software

The software that produced the partitions is denoted as follows

Note that a certain amount of care needs to be taken with the attribution of a result to a particular method. For example the finan512 graph with P = 4 is attributed to CHACO 2.0 although an equivalent

Abbreviation	Software	Reference
Grdy	Farhat's Greedy algorithm (as implemented within JOSTLE)	[Far88]
RCB	Recursive Coordinate Bisection (as implemented within JOSTLE)	[Sim91]
MRSB	Barnard & Simon's Multilevel Recursive Spectral Bisection	[BS94]
Ch2.0	CHACO - multilevel Kernighan-Lin (recursive bisection) ; version 2.0 (October 1995)	[HL95]
pM4.0	p-METIS - multilevel Kernighan-Lin (recursive bisection) ; available as part of METIS version 4.0 (September 1998)	[KK98a]
kM4.0	k-METIS - multilevel Kernighan-Lin (k-way) ; available as part of METIS version 4.0 (September 1998)	[KK98b]
GTS	Stephane Popinet's multilevel implementation available as part of the GNU Triangulated Surface Library	
J2.2	JOSTLE - multilevel Kernighan-Lin (k-way) ; version 2.2 (March 2000)	[WC00]
iJ	iterated JOSTLE - iterated multilevel Kernighan-Lin (k-way)	[Wal04]
JE	JOSTLE Evolutionary - combined evolutionary/multilevel scheme	[SWC04]
GrPart	Alexander Kozhushkin's implementation of iterative multilevel Kernighan-Lin	
MLSATS	MultiLevel refinated Mixed Simulated Annealing and Tabu Search	[BGOM03]
AMG	A bisection algorithm based on classical Algebraic MultiGrid from S. Wishko, A. Brandt and D. Ron	
MQI	Max-flow Quotient-cut Improvement, a bisection algorithm from K. Lang & S. Rao, which uses many multiple tries and improves an initial partition provided by METIS	[LR04]
SDP	A semi-definite programming approach together with a max-flow algorithm from K. Lang & S. Rao	
mpM4.0	Multiple runs (6561) of a randomised version of p-Metis ; results provided by K. Lang & S. Rao	
AES	ANTS, a multiagent algorithm, from Francesc Comellas & Emili Sapena	[CS05]
JKM	Some new matching heuristics from Jan Kasper Martinsen & Xing Cai	
GCSVD	Gene Correlation with Singular Value Decomposition from Jacob Martin	[Mar05]
PROBE	A PROBE based heuristic from P. Chardaire, M. Barake, and G. P. McKeown	[CBM07]
N/A	not available/recorded (probably JOSTLE Evolutionary)	

quality partition was also found by p-METIS, k-METIS, JOSTLE 2.2, JOSTLE Evolutionary and iterated JOSTLE. Attribution is thus given to the earliest method which found it but does not imply that an equivalent quality partition has not been found by a more recent method. The results were added into the archive in the order shown in the table (which is approximately the order in which the methods and software were developed).

Notes :

- CHACO was run using the multilevel KL method with recursive bisection and a coarsening threshold of 200.
- JOSTLE was run with the imbalance threshold set to 0%, 1%, 3% (default) and 5% (i.e. 4 different tests for each graph and value of P).
- JOSTLE Evolutionary (JE) was run with the imbalance threshold set to 0% and 3% (i.e. 2 different tests for each graph and value of P). It was only tested in full on a subset of the graphs (add20, 3elt, uk, add32, whitaker3, crack, wing_nodal, fe_4elt2, vibrobox, 4elt, fe_sphere, cti, memplus, cs4, fe_pwt, bcsstk32, t60k, wing, brack2) although occasional JE attributions may be found for other graphs.

References

BGOM03 R. Banos, C. Gil, J. Ortega, and F. G. Montoya. Multilevel Heuristic Algorithm for Graph Partitioning. In *3rd European Workshop on Evolutionary Computation in Combinatorial Optimization*, volume 2611 of *LNCS*, pages 143-153. Springer, Berlin, 2003.

BS94 S. T. Barnard and H. D. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *Concurrency : Practice & Experience*, 6(2) :101-117, 1994.

CBM07 P. Chardaire, M. Barake, and G. P. McKeown. A PROBE based heuristic for Graph Partitioning. (Submitted to IEEE Trans. Comp.), 2007.

CS05 F. Comellas and E. Sapena. A multiagent algorithm for graph partitioning. Tech. Rep. UPC-MA4-2005-05C-21, Univ. Politècnica de Catalunya, 2005.

Far88 C. Farhat. A Simple and Efficient Automatic FEM Domain Decomposer. *Comput. & Structures*, 28(5) :579-602, 1988.

HK00 B. Hendrickson and T. G. Kolda. Graph Partitioning Models for Parallel Computing. *Parallel Comput.*, 26(12) :1519-1534, 2000.

HL95 B. Hendrickson and R. Leland. A Multilevel Algorithm for Partitioning Graphs. In S. Karin, editor, *Proc. Supercomputing '95, San Diego*. ACM Press, New York, 1995.

KK98a G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.*, 20(1) :359-392, 1998.

KK98b G. Karypis and V. Kumar. Multilevel k-way Partitioning Scheme for Irregular Graphs. *J. Parallel Distrib. Comput.*, 48(1) :96-129, 1998.

LR04 K. Lang and S. Rao. A flow-based method for improving the expansion or conductance of graph cuts. 2004.

Mar05 J. Martin. Subproblem Optimization by Gene Correlation with Singular Value Decomposition. In *Proc. Genetic & Evolutionary Comput. Conf. (GECCO-2005)*. ACM, 2005.

Sim91 H. D. Simon. Partitioning of Unstructured Problems for Parallel Processing. *Computing Systems Engrg.*, 2 :135-148, 1991.

graph	$\|V\|$	$\|E\|$	original source
add20	2395	7462	http ://www.cise.ufl.edu/research/sparse/matrices/Hamm/add20.html
data	2851	15093	http ://staffweb.cms.gre.ac.uk/ c.walshaw/partition/archive/data/data.graph
3elt	4720	13722	(ftp ://riacs.edu/pub/grids/3elt.grid.gz)
uk	4824	6837	http ://staffweb.cms.gre.ac.uk/ c.walshaw/partition/archive/uk/uk.graph
add32	4960	9462	http ://www.cise.ufl.edu/research/sparse/matrices/Hamm/add32.html
bcsstk33	8738	291583	http ://www.cise.ufl.edu/research/sparse/matrices/HB/bcsstk33.html
whitaker3	9800	28989	(ftp ://riacs.edu/pub/grids/whitaker3.grid.gz)
crack	10240	30380	wwwcs.uni-paderborn.de/fachbereich/AG/monien/RESEARCH/PART/GRAPHS/FEM2.tar
wing_nodal	10937	75488	http ://staffweb.cms.gre.ac.uk/ c.walshaw/partition/archive/wing_nodal/wing_nodal.graph
fe_4elt2	11143	32818	(ftp ://ftp.u-bordeaux.fr/pub/Local/Info/Software/Scotch/Graphs/fe_4elt2.src.gz
vibrobox	12328	165250	http ://www.cise.ufl.edu/research/sparse/matrices/Cote/vibrobox.html
bcsstk29	13992	302748	http ://www.cise.ufl.edu/research/sparse/matrices/HB/bcsstk29.html
4elt	15606	45878	(ftp ://riacs.edu/pub/grids/barth5.grid.gz)
fe_sphere	16386	49152	(ftp ://ftp.u-bordeaux.fr/pub/Local/Info/Software/Scotch/Graphs/fe_sphere.src.gz)
cti	16840	48232	http ://staffweb.cms.gre.ac.uk/ c.walshaw/partition/archive/cti/cti.graph
memplus	17758	54196	http ://www.cise.ufl.edu/research/sparse/matrices/Hamm/memplus.html
cs4	22499	43858	http ://staffweb.cms.gre.ac.uk/ c.walshaw/partition/archive/cs4/cs4.graph
bcsstk30	28924	1007284	http ://www.cise.ufl.edu/research/sparse/matrices/HB/bcsstk30.html
bcsstk31	35588	572914	http ://www.cise.ufl.edu/research/sparse/matrices/HB/bcsstk31.html
fe_pwt	36519	144794	(ftp ://ftp.u-bordeaux.fr/pub/Local/Info/Software/Scotch/Graphs/fe_pwt.src.gz)
bcsstk32	44609	985046	http ://www.cise.ufl.edu/research/sparse/matrices/HB/bcsstk32.html
fe_body	45087	163734	(ftp ://ftp.u-bordeaux.fr/pub/Local/Info/Software/Scotch/Graphs/fe_body.src.gz)
t60k	60005	89440	http ://staffweb.cms.gre.ac.uk/ c.walshaw/partition/archive/t60k/t60k.graph
wing	62032	121544	http ://staffweb.cms.gre.ac.uk/ c.walshaw/partition/archive/wing/wing.graph
brack2	62631	366559	(ftp ://riacs.edu/pub/grids/brack2.grid.gz)
finan512	74752	261120	http ://www.cise.ufl.edu/research/sparse/matrices/Mulvey/finan512.html
fe_tooth	78136	452591	(ftp ://ftp.u-bordeaux.fr/pub/Local/Info/Software/Scotch/Graphs/fe_tooth.src.gz)
fe_rotor	99617	662431	(ftp ://ftp.u-bordeaux.fr/pub/Local/Info/Software/Scotch/Graphs/fe_rotor.src.gz)
598a	110971	741934	(ftp ://ftp.cs.umn.edu/users/kumar/Graphs/598a.graph.gz)
fe_ocean	143437	409593	(ftp ://ftp.u-bordeaux.fr/pub/Local/Info/Software/Scotch/Graphs/fe_ocean.src.gz)
144	144649	1074393	(ftp ://ftp.cs.umn.edu/users/kumar/Graphs/144.graph.gz)
wave	156317	1059331	(ftp ://riacs.edu/pub/grids/wave.grid.gz)
m14b	214765	1679018	(ftp ://ftp.cs.umn.edu/users/kumar/Graphs/m14b.graph.gz)
auto	448695	3314611	(ftp ://ftp.cs.umn.edu/users/kumar/Graphs/auto.graph.gz)

ST97 H. D. Simon and S.-H. Teng. How Good is Recursive Bisection ? *SIAM J. Sci. Comput.*, 18(5) :1436-1445, 1997.

SWC04 A. J. Soper, C. Walshaw, and M. Cross. A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph Partitioning. *J. Global Optimization*, 29(2) :225-241, 2004.

Wal04 C. Walshaw. Multilevel Refinement for Combinatorial Optimisation Problems. *Annals Oper. Res.*, 131 :325-372, 2004.

WC00 C. Walshaw and M. Cross. Mesh Partitioning : a Multilevel Balancing and Refinement Algorithm. *SIAM J. Sci. Comput.*, 22(1) :63-80, 2000.

Test Graphs

The test graphs are taken from various sources and listed in the table below. Graphs for which the original source is bracketted appear to be no longer available at that site. However all the graph files are also available from this site converted into a standard format. This format is the same as used by JOSTLE, CHACO & METIS and is described in the JOSTLE userguide.

graph	2	4	8	16	32	64
add20	596 (1198) [PROBE]	1203 (599) [JE]	1758 (300) [iJ]	2216 (150) [iJ]	2765 (75) [JE]	3266 (38) [JE]
data	189 (1426) [PROBE]	429 (713) [iJ]	728 (357) [iJ]	1245 (179) [Ch2.0]	2004 (90) [iJ]	3016 (45) [iJ]
3elt	90 (2360) [JE]	201 (1180) [JE]	349 (590) [JE]	589 (295) [JE]	972 (148) [JE]	1594 (74) [iJ]
uk	20 (2412) [mpM4.0]	44 (1206) [JE]	91 (603) [JE]	162 (302) [JE]	286 (151) [JE]	456 (76) [iJ]
add32	11 (2480) [Ch2.0]	37 (1240) [pM4.0]	75 (620) [JE]	121 (310) [Ch2.0]	234 (155) [JE]	720 (78) [Ch2.0]
bcsstk33	10171 (4369) [GCSVD]	21956 (2185) [iJ]	35088 (1093) [iJ]	56973 (547) [iJ]	80099 (274) [iJ]	109585 (137) [iJ]
whitaker3	127 (4900) [JE]	382 (2450) [JE]	664 (1225) [JE]	1110 (613) [JE]	1719 (307) [JE]	2579 (154) [iJ]
crack	184 (5120) [JE]	368 (2560) [JE]	687 (1280) [JE]	1124 (640) [JE]	1768 (320) [JE]	2696 (160) [iJ]
wing_nodal	1707 (5469) [JE]	3581 (2735) [JE]	5443 (1368) [JE]	8422 (684) [JE]	12228 (342) [iJ]	16373 (171) [iJ]
fe_4elt2	130 (5572) [MRSB]	349 (2786) [JE]	617 (1393) [JE]	1028 (697) [JE]	1677 (349) [JE]	2537 (175) [iJ]
vibrobox	10343 (6164) [JE]	19245 (3082) [JE]	25468 (1541) [iJ]	33468 (771) [iJ]	43786 (386) [JE]	51101 (193) [JE]
bcsstk29	2843 (6996) [mpM4.0]	8786 (3498) [MRSB]	16832 (1749) [iJ]	24706 (875) [iJ]	38535 (438) [iJ]	61252 (219) [iJ]
4elt	139 (7803) [JE]	327 (3902) [JE]	556 (1951) [JE]	968 (976) [JE]	1606 (488) [JE]	2702 (244) [iJ]
fe_sphere	386 (8193) [JE]	773 (4097) [N/A]	1193 (2049) [JE]	1750 (1025) [JE]	2567 (513) [JE]	3663 (257) [JE]
cti	334 (8420) [JE]	977 (4210) [JE]	1812 (2105) [JE]	2915 (1053) [JE]	4460 (527) [iJ]	6277 (264) [iJ]
memplus	5513 (8879) [PROBE]	9754 (4440) [JE]	12095 (2220) [iJ]	13742 (1110) [JE]	14830 (555) [iJ]	17446 (278) [JE]
cs4	372 (11250) [JE]	964 (5625) [JE]	1496 (2813) [JE]	2206 (1407) [JE]	3110 (704) [iJ]	4223 (352) [iJ]
bcsstk30	6394 (14462) [JE]	16652 (7231) [JE]	34921 (3616) [JE]	72007 (1808) [JE]	120580 (904) [JE]	184855 (452) [JE]
bcsstk31	2762 (17794) [PROBE]	8322 (8897) [iJ]	14426 (4449) [pM4.0]	25491 (2225) [iJ]	39440 (1113) [iJ]	63998 (557) [iJ]
fe_pwt	340 (18260) [GrPart]	709 (9130) [JE]	1465 (4565) [JE]	2855 (2283) [JE]	5848 (1142) [iJ]	8495 (571) [iJ]
bcsstk32	4667 (22305) [JE]	10578 (11153) [JE]	22826 (5577) [JE]	40275 (2789) [JE]	66407 (1395) [JE]	102820 (698) [JE]
fe_body	262 (22544) [MQI]	703 (11272) [pM4.0]	1315 (5636) [iJ]	2117 (2818) [iJ]	3432 (1409) [iJ]	5588 (705) [iJ]
t60k	80 (30003) [JE]	216 (15002) [JE]	480 (7501) [JE]	892 (3751) [iJ]	1453 (1876) [iJ]	2277 (938) [iJ]
wing	791 (31016) [MQI]	1666 (15508) [JE]	2589 (7754) [JE]	4198 (3877) [JE]	6104 (1939) [iJ]	8203 (970) [iJ]
brack2	731 (31316) [JE]	3090 (15658) [JE]	7269 (7829) [JE]	12324 (3915) [JE]	18421 (1958) [iJ]	27881 (979) [iJ]
finan512	162 (37376) [Ch2.0]	324 (18688) [Ch2.0]	648 (9344) [Ch2.0]	1296 (4672) [Ch2.0]	2592 (2336) [Ch2.0]	11192 (1168) [pM4.0]
fe_tooth	3850 (39068) [mpM4.0]	7605 (19534) [iJ]	12850 (9767) [iJ]	18804 (4884) [iJ]	26642 (2442) [iJ]	36766 (1221) [iJ]
fe_rotor	2098 (49809) [PROBE]	8097 (24905) [Ch2.0]	14953 (12453) [Ch2.0]	21241 (6227) [iJ]	34320 (3114) [iJ]	52626 (1557) [pM4.0]
598a	2398 (55486) [PROBE]	8379 (27743) [Ch2.0]	17223 (13872) [Ch2.0]	28152 (6936) [iJ]	42088 (3468) [iJ]	59708 (1734) [iJ]
fe_ocean	464 (71719) [PROBE]	1923 (35860) [JE]	4760 (17930) [N/A]	8622 (8965) [iJ]	14277 (4483) [iJ]	22301 (2242) [iJ]
144	6490 (72325) [PROBE]	15964 (36163) [N/A]	27751 (18082) [N/A]	39856 (9041) [N/A]	58659 (4521) [iJ]	82123 (2261) [iJ]
wave	8692 (78159) [PROBE]	18774 (39080) [iJ]	32204 (19540) [iJ]	47277 (9770) [iJ]	66891 (4885) [iJ]	91784 (2443) [iJ]
m14b	3836 (107383) [MQI]	14013 (53692) [Ch2.0]	28128 (26846) [pM4.0]	45362 (13423) [iJ]	68468 (6712) [N/A]	103946 (3356) [iJ]
auto	10117 (224348) [PROBE]	29409 (112174) [iJ]	53326 (56087) [pM4.0]	83748 (28044) [iJ]	131236 (14022) [iJ]	182934 (7011) [iJ]

TAB. A.1 – Results with up to 0% imbalance, $S\_max <= 1.00 * S\_opt$

graph	2	4	8	16	32	64
add20	594 (1209) [GCSVD]	1203 (599) [JE]	1758 (300) [iJ]	2216 (150) [iJ]	2765 (75) [JE]	3266 (38) [JE]
data	188 (1438) [AMG]	425 (717) [iJ]	719 (358) [iJ]	1245 (179) [Ch2.0]	2004 (90) [iJ]	3016 (45) [iJ]
3elt	89 (2382) [GrPart]	199 (1186) [JE]	349 (590) [JE]	589 (295) [JE]	972 (148) [JE]	1594 (74) [iJ]
uk	19 (2414) [AMG]	44 (1206) [JE]	86 (608) [JE]	157 (305) [JE]	285 (152) [iJ]	456 (76) [iJ]
add32	10 (2481) [J2.2]	33 (1241) [JE]	69 (621) [JE]	117 (311) [JE]	212 (156) [JE]	720 (78) [Ch2.0]
bcsstk33	10109 (4407) [GrPart]	21685 (2203) [iJ]	35088 (1093) [iJ]	55546 (551) [iJ]	79121 (275) [iJ]	109585 (137) [iJ]
whitaker3	126 (4908) [JE]	380 (2458) [JE]	660 (1231) [JE]	1110 (613) [JE]	1719 (307) [JE]	2579 (154) [iJ]
crack	183 (5161) [AMG]	368 (2560) [JE]	678 (1291) [N/A]	1124 (640) [JE]	1707 (323) [iJ]	2623 (161) [iJ]
wing_nodal	1696 (5522) [MQI]	3572 (2747) [JE]	5443 (1368) [JE]	8422 (684) [JE]	12228 (342) [iJ]	16221 (172) [iJ]
fe_4elt2	130 (5572) [MRSB]	349 (2786) [JE]	611 (1402) [iJ]	1028 (697) [JE]	1677 (349) [JE]	2537 (175) [iJ]
vibrobox	10310 (6184) [JE]	19245 (3082) [JE]	25001 (1556) [iJ]	33468 (771) [iJ]	43415 (388) [JE]	50849 (194) [JE]
bcsstk29	2818 (7008) [GrPart]	8786 (3498) [MRSB]	16505 (1758) [iJ]	24706 (875) [iJ]	37810 (441) [iJ]	60795 (220) [iJ]
4elt	138 (7844) [GrPart]	327 (3902) [JE]	556 (1951) [JE]	961 (985) [iJ]	1606 (488) [JE]	2689 (246) [iJ]
fe_sphere	386 (8193) [JE]	768 (4098) [N/A]	1152 (2060) [JE]	1730 (1033) [iJ]	2565 (516) [iJ]	3663 (257) [JE]
cti	318 (8480) [JE]	967 (4247) [N/A]	1812 (2105) [JE]	2915 (1053) [JE]	4460 (527) [iJ]	6125 (265) [iJ]
memplus	5492 (8937) [JE]	9754 (4440) [JE]	11939 (2241) [iJ]	13742 (1110) [JE]	14830 (555) [iJ]	17446 (278) [JE]
cs4	367 (11351) [JE]	940 (5638) [JE]	1476 (2840) [N/A]	2206 (1407) [JE]	3110 (704) [iJ]	4223 (352) [iJ]
bcsstk30	6335 (14603) [GrPart]	16622 (7301) [N/A]	34604 (3649) [JE]	72007 (1808) [JE]	120468 (912) [iJ]	180470 (456) [iJ]
bcsstk31	2701 (17965) [GrPart]	7879 (8898) [pM4.0]	14426 (4449) [pM4.0]	25491 (2225) [iJ]	39440 (1113) [iJ]	63626 (561) [iJ]
fe_pwt	340 (18260) [GrPart]	705 (9139) [JE]	1442 (4583) [JE]	2855 (2283) [JE]	5779 (1152) [iJ]	8454 (576) [iJ]
bcsstk32	4667 (22305) [JE]	10578 (11153) [JE]	22826 (5577) [JE]	39682 (2790) [JE]	65389 (1407) [iJ]	102820 (698) [JE]
fe_body	262 (22544) [MQI]	703 (11272) [pM4.0]	1197 (5637) [pM4.0]	1981 (2819) [pM4.0]	3370 (1423) [iJ]	5518 (711) [iJ]
t60k	77 (30299) [AMG]	213 (15027) [iJ]	467 (7534) [N/A]	878 (3783) [N/A]	1450 (1892) [iJ]	2277 (938) [iJ]
wing	787 (31278) [MQI]	1666 (15508) [JE]	2589 (7754) [JE]	4198 (3877) [JE]	6074 (1957) [iJ]	8161 (978) [iJ]
brack2	708 (31614) [GrPart]	3090 (15658) [JE]	7269 (7829) [JE]	12324 (3915) [JE]	18421 (1958) [iJ]	27881 (979) [iJ]
finan512	162 (37376) [Ch2.0]	324 (18688) [Ch2.0]	648 (9344) [Ch2.0]	1296 (4672) [Ch2.0]	2592 (2336) [Ch2.0]	11192 (1168) [pM4.0]
fe_tooth	3823 (39456) [SDP]	7435 (19725) [iJ]	12850 (9767) [iJ]	18435 (4928) [iJ]	26642 (2442) [iJ]	36487 (1233) [iJ]
fe_rotor	2049 (50305) [GrPart]	8097 (24905) [Ch2.0]	14028 (12576) [iJ]	20773 (6288) [iJ]	33686 (3144) [iJ]	50912 (1572) [iJ]
598a	2388 (55856) [MQI]	8311 (27985) [N/A]	16594 (14010) [iJ]	27344 (7005) [iJ]	41124 (3502) [iJ]	59152 (1751) [iJ]
fe_ocean	387 (72425) [GrPart]	1911 (36195) [N/A]	4760 (17930) [N/A]	8507 (9049) [iJ]	13767 (4526) [iJ]	21854 (2263) [iJ]
144	6479 (72980) [MQI]	15345 (36366) [N/A]	25818 (18201) [N/A]	39856 (9041) [N/A]	58659 (4521) [iJ]	81145 (2268) [N/A]
wave	8682 (78934) [MQI]	18058 (39395) [iJ]	32204 (19540) [iJ]	46903 (9860) [iJ]	65926 (4933) [iJ]	91304 (2466) [iJ]
m14b	3826 (107831) [MQI]	13844 (53971) [JKM]	28128 (26846) [pM4.0]	44515 (13433) [N/A]	68468 (6712) [N/A]	101980 (3382) [N/A]
auto	10042 (226571) [SDP]	29158 (112676) [iJ]	53158 (56646) [iJ]	82901 (28256) [iJ]	130818 (14160) [iJ]	182890 (7079) [iJ]

TAB. A.2 – Results with up to 1% imbalance, $S\_max <= 1.01 * S\_opt$

graph	2	4	8	16	32	64
add20	576 (1233) [GCSVD]	1188 (616) [JE]	1720 (308) [iJ]	2216 (150) [iJ]	2758 (76) [JE]	3266 (38) [JE]
data	185 (1465) [AMG]	383 (733) [iJ]	708 (367) [iJ]	1195 (183) [Ch2.0]	1970 (91) [iJ]	2911 (46) [iJ]
3elt	87 (2398) [JE]	199 (1186) [JE]	336 (607) [JE]	566 (303) [JE]	958 (151) [N/A]	1552 (75) [N/A]
uk	18 (2455) [JE]	41 (1228) [JE]	82 (620) [JE]	154 (308) [N/A]	265 (155) [N/A]	455 (77) [iJ]
add32	10 (2481) [J2.2]	33 (1241) [JE]	69 (621) [JE]	117 (311) [JE]	212 (156) [JE]	624 (80) [pM4.0]
bcsstk33	10064 (4419) [AMG]	21423 (2250) [iJ]	34322 (1124) [iJ]	55546 (551) [iJ]	79121 (275) [iJ]	109348 (140) [iJ]
whitaker3	126 (4908) [JE]	380 (2458) [JE]	658 (1261) [JE]	1092 (630) [JE]	1686 (315) [JE]	2535 (157) [JE]
crack	182 (5224) [AMG]	360 (2616) [JE]	676 (1309) [JE]	1082 (659) [JE]	1679 (329) [N/A]	2592 (164) [JE]
wing_nodal	1680 (5629) [SDP]	3566 (2796) [JE]	5401 (1407) [JE]	8316 (704) [JE]	12024 (352) [JE]	16221 (172) [iJ]
fe_4elt2	130 (5572) [MRSB]	349 (2786) [JE]	603 (1429) [JE]	1007 (717) [JE]	1651 (358) [JE]	2537 (175) [iJ]
vibrobox	10310 (6184) [JE]	19245 (3082) [JE]	24874 (1587) [JE]	32002 (793) [iJ]	42582 (396) [iJ]	50849 (194) [JE]
bcsstk29	2818 (7008) [GrPart]	8541 (3600) [J2.2]	16505 (1758) [iJ]	24706 (875) [iJ]	37810 (441) [iJ]	58452 (225) [iJ]
4elt	137 (8017) [JE]	319 (3966) [N/A]	527 (2009) [N/A]	916 (1004) [JE]	1537 (502) [JE]	2581 (251) [JE]
fe_sphere	384 (8289) [JE]	766 (4155) [JE]	1152 (2060) [JE]	1706 (1053) [N/A]	2477 (527) [JE]	3547 (263) [JE]
cti	318 (8480) [JE]	917 (4331) [JE]	1716 (2160) [JE]	2778 (1080) [JE]	4236 (542) [iJ]	5992 (271) [iJ]
memplus	5407 (9140) [JE]	9664 (4542) [JE]	11939 (2241) [iJ]	13559 (1143) [JE]	14548 (571) [iJ]	17409 (285) [JE]
cs4	362 (11586) [AMG]	936 (5704) [JE]	1472 (2861) [N/A]	2126 (1448) [N/A]	3110 (704) [iJ]	4196 (362) [iJ]
bcsstk30	6251 (14679) [JE]	16617 (7341) [JE]	34559 (3666) [JE]	70768 (1851) [JE]	117232 (930) [JE]	178578 (465) [JE]
bcsstk31	2676 (18184) [MLSATS]	7879 (8898) [pM4.0]	14426 (4449) [pM4.0]	25160 (2290) [iJ]	38572 (1145) [iJ]	61066 (572) [iJ]
fe_pwt	340 (18260) [GrPart]	704 (9365) [JE]	1436 (4701) [JE]	2784 (2338) [JE]	5606 (1175) [JE]	8346 (587) [iJ]
bcsstk32	4667 (22305) [JE]	9728 (11286) [JE]	21307 (5741) [JE]	38929 (2864) [JE]	64302 (1435) [iJ]	96168 (717) [iJ]
fe_body	262 (22544) [MQI]	703 (11272) [pM4.0]	1197 (5637) [pM4.0]	1981 (2819) [pM4.0]	3223 (1451) [iJ]	5502 (725) [iJ]
t60k	71 (30795) [AMG]	211 (15219) [iJ]	467 (7534) [N/A]	852 (3862) [N/A]	1420 (1922) [iJ]	2258 (958) [iJ]
wing	774 (31824) [MQI]	1636 (15815) [JE]	2551 (7894) [JE]	4015 (3966) [JE]	6028 (1996) [JE]	8161 (978) [iJ]
brack2	684 (32168) [JE]	2864 (16127) [JE]	7080 (8057) [JE]	11958 (4030) [JE]	17952 (2015) [JE]	27345 (1007) [iJ]
finan512	162 (37376) [Ch2.0]	324 (18688) [Ch2.0]	648 (9344) [Ch2.0]	1296 (4672) [Ch2.0]	2592 (2336) [Ch2.0]	10821 (1203) [N/A]
fe_tooth	3792 (40210) [SDP]	7435 (19725) [iJ]	12850 (9767) [iJ]	18435 (4928) [iJ]	26642 (2442) [iJ]	36030 (1257) [iJ]
fe_rotor	1965 (51065) [SDP]	8097 (24905) [Ch2.0]	14028 (12576) [iJ]	20773 (6288) [iJ]	33686 (3144) [iJ]	47972 (1603) [iJ]
598a	2367 (56705) [MQI]	7978 (28520) [N/A]	16031 (14286) [N/A]	26257 (7143) [N/A]	40792 (3571) [N/A]	58454 (1784) [iJ]
fe_ocean	311 (73322) [GrPart]	1704 (36802) [N/A]	4019 (18436) [N/A]	7838 (9223) [N/A]	12746 (4616) [N/A]	21854 (2263) [iJ]
144	6438 (74281) [MQI]	15250 (37037) [N/A]	25611 (18523) [N/A]	38478 (9311) [N/A]	58659 (4521) [iJ]	80894 (2287) [N/A]
wave	8616 (80306) [SDP]	18058 (39395) [iJ]	30895 (20087) [iJ]	46487 (10056) [iJ]	64953 (5027) [iJ]	88383 (2515) [iJ]
m14b	3823 (108879) [MQI]	13844 (53971) [JKM]	27711 (27431) [iJ]	44174 (13785) [iJ]	68468 (6712) [N/A]	101980 (3382) [N/A]
auto	9782 (230275) [SDP]	29158 (112676) [iJ]	48329 (57712) [JKM]	82901 (28256) [iJ]	130310 (14437) [iJ]	180562 (7211) [iJ]

TAB. A.3 – Results with up to 3% imbalance, $S\_max <= 1.03 * S\_opt$

graph	2	4	8	16	32	64
add20	550 (1256) [GCSVD]	1184 (628) [iJ]	1705 (314) [iJ]	2186 (157) [MLSATS]	2758 (76) [JE]	3266 (38) [JE]
data	181 (1497) [SDP]	378 (746) [iJ]	702 (374) [iJ]	1195 (183) [Ch2.0]	1922 (93) [iJ]	2911 (46) [iJ]
3elt	87 (2398) [JE]	199 (1186) [JE]	334 (617) [iJ]	566 (303) [JE]	958 (151) [N/A]	1552 (75) [N/A]
uk	18 (2455) [JE]	41 (1228) [JE]	82 (620) [JE]	154 (308) [N/A]	265 (155) [N/A]	436 (79) [iJ]
add32	10 (2481) [J2.2]	33 (1241) [JE]	69 (621) [JE]	117 (311) [JE]	212 (156) [JE]	624 (80) [pM4.0]
bcsstk33	9914 (4554) [iJ]	21229 (2293) [iJ]	34117 (1145) [iJ]	55546 (551) [iJ]	78821 (286) [iJ]	108249 (143) [iJ]
whitaker3	126 (4908) [JE]	380 (2458) [JE]	658 (1261) [JE]	1092 (630) [JE]	1686 (315) [JE]	2535 (157) [JE]
crack	182 (5224) [AMG]	360 (2616) [JE]	676 (1309) [JE]	1082 (659) [JE]	1679 (329) [N/A]	2590 (168) [iJ]
wing_nodal	1668 (5742) [SDP]	3566 (2796) [JE]	5387 (1429) [MLSATS]	8316 (704) [JE]	12024 (352) [JE]	16102 (179) [iJ]
fe_4elt2	130 (5572) [MRSB]	349 (2786) [JE]	597 (1457) [iJ]	1007 (717) [JE]	1651 (358) [JE]	2516 (182) [iJ]
vibrobox	10310 (6184) [JE]	19245 (3082) [JE]	24158 (1603) [iJ]	31695 (809) [iJ]	41176 (404) [iJ]	50757 (202) [iJ]
bcsstk29	2818 (7008) [GrPart]	8088 (3672) [MLSATS]	15314 (1836) [MLSATS]	24706 (875) [iJ]	36731 (456) [iJ]	58108 (229) [iJ]
4elt	137 (8017) [JE]	319 (3966) [N/A]	527 (2009) [N/A]	916 (1004) [JE]	1537 (502) [JE]	2581 (251) [JE]
fe_sphere	384 (8289) [JE]	766 (4155) [JE]	1152 (2060) [JE]	1692 (1074) [iJ]	2477 (527) [JE]	3547 (263) [JE]
cti	318 (8480) [JE]	917 (4331) [JE]	1716 (2160) [JE]	2778 (1080) [JE]	4236 (542) [iJ]	5907 (276) [iJ]
memplus	5353 (9322) [iJ]	9427 (4652) [MLSATS]	11939 (2241) [iJ]	13279 (1165) [iJ]	14384 (582) [iJ]	17409 (285) [JE]
cs4	356 (11794) [AMG]	936 (5704) [JE]	1472 (2861) [N/A]	2126 (1448) [N/A]	3080 (737) [iJ]	4196 (362) [iJ]
bcsstk30	6251 (14679) [JE]	16617 (7341) [JE]	34559 (3666) [JE]	70768 (1851) [JE]	117232 (930) [JE]	177379 (474) [iJ]
bcsstk31	2676 (18184) [MLSATS]	7879 (8898) [pM4.0]	13561 (4670) [iJ]	24179 (2331) [iJ]	38572 (1145) [iJ]	60446 (583) [iJ]
fe_pwt	340 (18260) [GrPart]	704 (9365) [JE]	1436 (4701) [JE]	2784 (2338) [JE]	5606 (1175) [JE]	8310 (599) [iJ]
bcsstk32	4667 (22305) [JE]	9728 (11286) [JE]	21307 (5741) [JE]	38320 (2923) [iJ]	62961 (1463) [iJ]	96168 (717) [iJ]
fe_body	262 (22544) [MQI]	703 (11272) [pM4.0]	1145 (5885) [iJ]	1833 (2952) [iJ]	3223 (1451) [iJ]	5219 (739) [iJ]
t60k	65 (31437) [SDP]	211 (15219) [iJ]	467 (7534) [N/A]	852 (3862) [N/A]	1420 (1922) [iJ]	2221 (984) [iJ]
wing	770 (32550) [MQI]	1636 (15815) [JE]	2551 (7894) [JE]	4015 (3966) [JE]	6010 (2031) [iJ]	8161 (978) [iJ]
brack2	660 (32600) [SDP]	2808 (16439) [MLSATS]	7080 (8057) [JE]	11958 (4030) [JE]	17952 (2015) [JE]	26944 (1027) [iJ]
finan512	162 (37376) [Ch2.0]	324 (18688) [Ch2.0]	648 (9344) [Ch2.0]	1296 (4672) [Ch2.0]	2592 (2336) [Ch2.0]	10821 (1203) [N/A]
fe_tooth	3773 (40567) [SDP]	7152 (20405) [MLSATS]	12646 (10071) [iJ]	18435 (4928) [iJ]	26016 (2563) [MLSATS]	36030 (1257) [iJ]
fe_rotor	1957 (51783) [SDP]	8097 (24905) [Ch2.0]	13184 (12999) [iJ]	20773 (6288) [iJ]	33686 (3144) [iJ]	47852 (1633) [iJ]
598a	2336 (57855) [MQI]	7978 (28520) [N/A]	16031 (14286) [N/A]	26257 (7143) [N/A]	40179 (3641) [iJ]	58307 (1820) [iJ]
fe_ocean	311 (73322) [GrPart]	1704 (36802) [N/A]	4019 (18436) [N/A]	7838 (9223) [N/A]	12746 (4616) [N/A]	21784 (2353) [iJ]
144	6362 (75917) [SDP]	15250 (37037) [N/A]	25611 (18523) [N/A]	38478 (9311) [N/A]	58142 (4701) [N/A]	80445 (2373) [N/A]
wave	8563 (81841) [MQI]	18058 (39395) [iJ]	30583 (20514) [MLSATS]	44625 (10258) [MLSATS]	63725 (5129) [iJ]	88383 (2515) [iJ]
m14b	3802 (112532) [MQI]	13844 (53971) [JKM]	27711 (27431) [iJ]	44174 (13785) [iJ]	68468 (6712) [N/A]	101385 (3523) [iJ]
auto	9450 (235532) [MQI]	29158 (112676) [iJ]	48221 (58885) [iJ]	82901 (28256) [iJ]	127433 (14722) [iJ]	179464 (7360) [iJ]

TAB. A.4 – Results with up to 5% imbalance, $S\_max <= 1.05 * S\_opt$

Annexe B

Compléments sur le système de gestion du trafic aérien

B.1 Les contrôleurs des centres de contrôle en route

Le contrôle en route est effectué par des contrôleurs regroupés en centres de contrôle. On distingue 5 centres de contrôle en route en France (Aix-en-Provence, Bordeaux, Brest, Paris et Reims), 36 en Europe, chaque centre pouvant gérer une ou plusieurs *zones de qualification* [1] (la France en compte 7). La localisation géographique de ces centres n'a pas de signification particulière par rapport aux zones qu'ils contrôlent (même si en France ces derniers sont plus ou moins situés au centre de leur zone d'action) mais correspond plutôt aux différentes politiques d'aménagement du territoire. Il existe encore peu de centres communs à plusieurs pays. Le centre suisse contrôle deux secteurs frontaliers avec la France, ceci grâce à un accord entre les deux pays. Il existe également deux centres « européens », c'est-à-dire gérés non plus par les pays dont ils contrôlent l'espace mais directement par l'agence Eurocontrol, agence regroupant divers pays européens dont le but est de développer et coordonner les différentes politiques de gestion de trafic aérien.

Les contrôleurs travaillent généralement par paire : un contrôleur *organique* et un contrôleur *tactique*. Chaque paire travaille devant une station de contrôle. Un exemple représentant une position occupée (ou *armée*) par deux contrôleurs est présenté figure B.1. La personne au premier plan étant le contrôleur organique, la seconde, à l'arrière, le contrôleur tactique.

Le contrôleur organique est en charge de la gestion du trafic aérien à moyen terme. Pour cela il dispose de *bandes de progression* ou, plus couramment, *strips* (originellement des bandelettes de papiers, parfois remplacées par un affichage électronique).

Les strips contiennent toutes les informations fondamentales concernant chaque vol passant dans le secteur dont il a la charge (un strip par vol). Par exemple, le strip présenté figure B.2 renseigne le contrôleur sur :

– L'identifiant du vol (AFR1715) et le type d'aéronef (A321),
– le niveau de vol d'entrée (FL310),
– le niveau de vol demandé (FL350),
– les aéroports de départ (LIMC, Milan) et d'arrivée (LFPG, Roissy-Charles de Gaulle),
– la fréquence radio de contact (132.1),

[1]Un contrôleur qualifié sur une zone géographique donnée ne peut pas contrôler sur une autre zone. Ainsi au sein d'un même centre de contrôle, un contrôleur ne peut pas travailler sur n'importe quelle position.

FIG. B.1 – Une position de contrôle

– et enfin la route prévue dans le plan de vol et l'heure de passage prévue sur chaque point de report (par exemple DIJ à 16h11, TRO à 16h23).

FIG. B.2 – Un strip électronique

Les informations contenues dans les strips sont transmises au contrôleur tactique une dizaine de minutes avant l'entrée de l'aéronef dans le secteur dont il a la charge. Avant cela le contrôleur organique est chargé de détecter tout conflit qui pourrait être induit par la future entrée d'un aéronef dans le secteur qu'il contrôle. Le contrôleur organique doit également assurer la liaison avec le contrôleur du secteur précédent et celui du secteur suivant afin d'assurer une bonne transmission des aéronefs. Cette procédure, appelée *coordination*, est une des tâches majeures du contrôle. Il assure aussi la gestion globale du flux dans le secteur et doit s'assurer que la tâche que devra effectuer le second contrôleur ne sera pas trop lourde.

Le contrôleur tactique a, quant à lui, la responsabilité à court terme du trafic et notamment la séparation des aéronefs entre eux. Pour cela, il doit maintenir en permanence une distance minimale entre les aéronefs. Cette séparation est d'environ 5 Nm (1 Nm = 1 819 m) dans le plan horizontal et 1 000 ft (1 ft = 30 cm) dans le plan vertical. Lorsque ces deux normes sont simultanément violées, on dit qu'il y a perte de séparation ou **conflit**. Pour faciliter le travail du contrôleur, les aéronefs volent, comme nous l'avons vu, sur des routes aériennes et lorsqu'ils sont stables, c'est-à-dire lorsque les aéronefs ne sont ni en montée, ni en descente, ils se fixent sur un niveau de vol « entier ». Ainsi un aéronef volera par exemple à 31 000 ft (niveau de vol FL310) mais pas à 31 700 ft. Les niveaux de vol standards sont ainsi séparés de 1 000 ft (FL290, FL300, FL310). Pour maintenir ces différentes séparations, le contrôleur tactique dispose d'une image radar lui montrant la direction de déplacement des aéronefs, leur vitesse respective, leur niveau de vol, etc. Un exemple est présenté figure B.3. Celle-ci ne montre qu'une partie de l'image complète que peut avoir un contrôleur. La partie gris clair correspond au secteur dont les contrôleurs ont la charge, la partie gris foncé permettant de voir les aéronefs entrer et sortir du secteur dont ils ont la charge. Il existe encore très peu d'aide à la détection et à la résolution de conflit. Le seul système automatique actif de détection aujourd'hui, du point de vue du contrôleur aérien, est le filet de sauvegarde, un dispositif qui se déclenche lorsque le risque de collision est imminent.

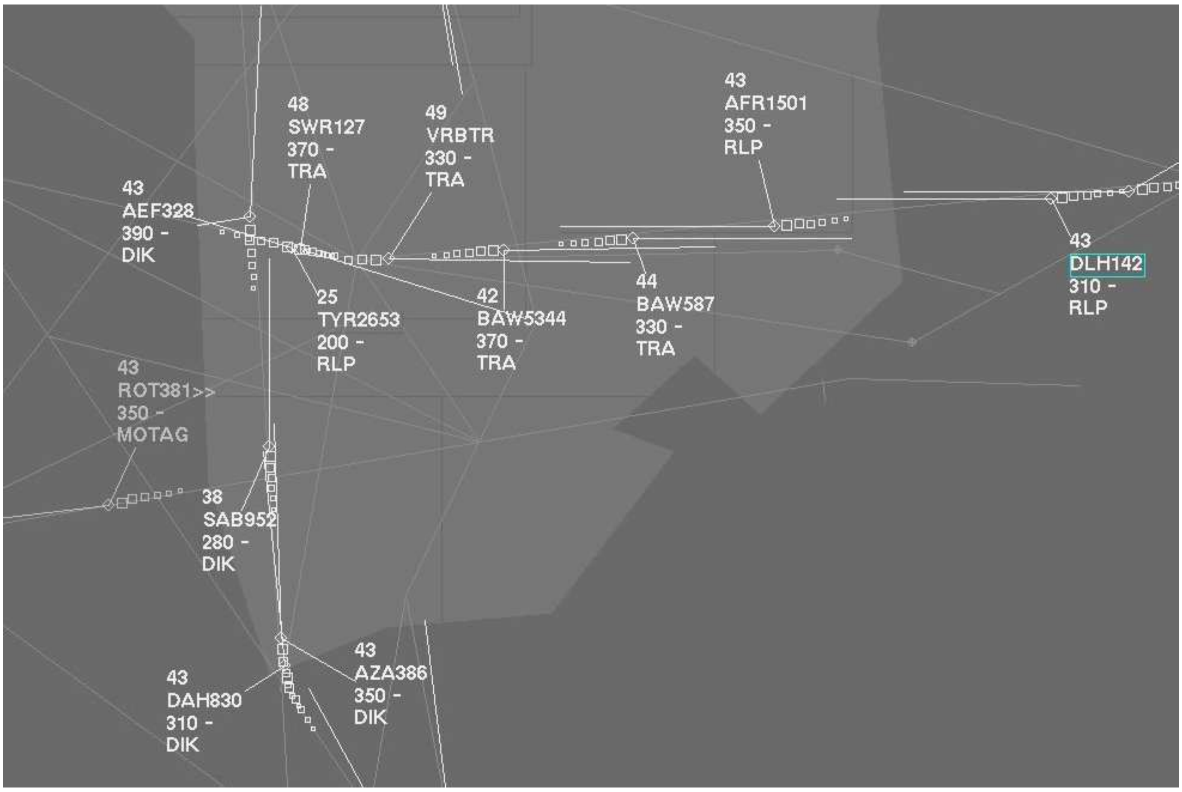


FIG. B.3 – Exemple d'image radar

Pour assurer la séparation, le contrôleur tactique dispose de deux catégories de manœuvres : les manœuvres en niveau et les manœuvres en cap. Les manœuvres en niveau se divisent elles-mêmes en trois catégories. Le contrôleur peut demander au pilote de :

– stabiliser son aéronef en montée ou en descente avant de l'autoriser à poursuivre vers son niveau initialement requis ;
– anticiper la descente lorsqu'il est proche de sa destination ;
– descendre [2] d'un niveau de vol lorsqu'il est stable en croisière (cette manœuvre n'étant pas du tout appréciée par les pilotes).

Les manœuvres en cap consistent simplement à modifier le cap d'un aéronef à gauche ou à droite,

[2] Il est également possible de demander à un aéronef de monter d'un niveau mais ceci dépend des performances avions et n'est pas toujours possible.

puis à lui faire reprendre sa trajectoire en lui communiquant un nouveau cap vers une balise de son plan de vol.

B.2 La régulation du trafic aérien

Nous avons vu précédemment que ce sont les pilotes qui choisissent leur plan de vol. Que se passe-t-il alors si tous les aéronefs veulent passer au même endroit au même moment ? Géré par des opérateurs humains, un secteur a une capacité limitée par la quantité d'aéronefs que peut gérer un couple de contrôleurs, capacité très inférieure à la capacité intrinsèque du secteur offert par son volume. Chaque secteur possède ainsi une capacité horaire dépendant du type de trafic (en route, en approche), de l'emplacement géographique du secteur, de sa taille, etc. Si cette capacité est dépassée, le contrôleur n'est plus à même de gérer tous les aéronefs et donc de garantir la sécurité des vols. Des méthodes ont ainsi été mises en place pour éviter de telles situations. C'est la gestion des flux de trafic (*Air Traffic Flow Management* ou ATFM).

La majorité des vols dépassant le cadre purement national (environ 15000 vols par jour sur les 25000 circulant en Europe lors d'une journée moyenne en 2005), la régulation du trafic doit se faire au niveau européen plutôt que national. En effet pour un vol venant du nord et atterrissant à Roissy-Charles de Gaulle, il est parfois nécessaire d'appliquer des mesures de régulation alors que l'aéronef est toujours pris en charge pas le contrôle belge ou britannique. En Europe c'est l'agence Eurocontrol qui est en charge de fournir les services ATFM à travers la CFMU (*Central Flow Management Unit*). La figure B.4 présente (en gris foncé) les pays européens couverts par la CFMU.

FIG. B.4 – Pays collaborant au travers de la CFMU

Pour décoller d'un aéroport, la CFMU attribue aux aéronefs un créneau de décollage, c'est-à-dire que ceux-ci doivent impérativement partir dans un laps de temps compris entre 5 minutes avant et jusqu'à 10 minutes après l'heure officielle de décollage. Si un aéronef manque son créneau, il ne peut, en règle générale, décoller et doit théoriquement en redemander un autre. En fonction des demandes de régulation, la CFMU choisit de modifier l'heure de décollage de certains vols. Elle impose donc une heure de départ différente de celle prévue à l'origine (Si un aéronef doit être décalé, le délai appliqué est donné à la compagnie aérienne 24 heures à l'avance).

L'algorithme, nommé CASA (pour *Computer Assisted Slot Allocation* [CFM00]), utilisé par la CFMU est très simple. Pour chaque zone régulée, il génère une liste de créneaux de passage. Ainsi, si un secteur d'une capacité horaire de 30 aéronefs par heure se trouve régulé, CASA va générer une liste de créneaux de passage espacés de deux minutes chacun (60 minutes divisées par la capacité) et cela pour toute la durée de la régulation.

La liste ainsi générée va se remplir au fur et à mesure que les compagnies aériennes soumettent leurs plans de vol à la CFMU. Si un aéronef « veut » passer dans un créneau déjà alloué à un autre vol c'est l'aéronef qui aurait dû passer le premier en l'absence de régulation qui obtient le créneau. L'autre aéronef est reclassé dans le créneau suivant. Si ce créneau est occupé alors le processus se répète. Il peut donc se produire une réaction en chaîne. Un nouveau vol peut décaler de proche en proche de nombreux autres.

Si un vol traverse plusieurs zones régulées il se voit imposer le retard le plus important. Il sera donc accepté avec ce retard dans toutes les zones. Le mécanisme employé ici par la CFMU n'est pas très clair. Illustrons ce qui se passe sur un exemple simple. Soit un aéronef qui traverse deux zones régulées. La première lui donne un retard de 10 minutes et la seconde un retard de 25 minutes. L'aéronef partira avec 25 minutes de retard. Mais le premier secteur peut-il prendre l'aéronef sans problème avec 25 minutes de retard au lieu de 10 ? Il n'existe pas de réponse officielle à ce problème.

Face aux problèmes d'incertitude sur les heures de passage et sur la capacité réelle des secteurs, la CFMU apporte une réponse pragmatique et statistique. L'ensemble des acteurs du transport aérien respecte les décisions prises et dans les faits les résultats sont acceptables. Toutefois, les fondements théoriques pour justifier son fonctionnement sont quasiment inexistants.

Glossaire

ATFM	Air Traffic Flow Management, 206
ATM	Air Traffic Management, 127
CASA	Computer Assisted Slot Allocation, 206
Centre	Centre de contrôle en route, organisme chargé du contrôle d'une ou plusieurs zones de qualification, 3, 116, 118–122, 124, 125, 135, 137, 138, 140, 143, 145, 149, 186
CFMU	Central Flow Management Unit, 206, 207
Ciel unique	Projet de ciel unique européen de la Commission européenne, 3, 120
Core area	Région européenne de plus forte densité de traffic aérien, 3, 118, 128, 130, 143, 145, 149, 186
DSNA	Direction des services de la navigation aérienne, 1, 121
ENAC	École nationale de l'aviation civile, 1
FAB	Fonctional Airspace Block, 3, 120–122, 128, 138, 140, 143, 149, 186
FMS	Flight Management Display, 114
GBP	Generalized Belief Propagation, 154–156
GGGP	Greedy graph growing Algorithm, 28, 185
GGP	Graph growing Algorithm, 28, 185
GKLR	Global Kernighan-Lin refinement, 41–43, 50, 106, 107, 164, 167, 168, 172, 185
GPS	Global Positionning System, 115
HEM	Heavy Edge Matching, 48
IFR	Instruments Flight Rules, 113, 126
ILS	Iterative Local Search, 44, 80

Kernighan-Lin	Algorithme inspiré de la notion de gain introduite par l'algorithme de Kernighan-Lin, 1, 5, 18, 25, 28, 33, 34, 36–38, 40–43, 45, 50, 53, 54, 64, 80, 101, 102, 106, 107, 133, 164, 191–193
Niveau de vol	Niveau de vol ou en anglais *Flight Level* (FL), 115, 116, 118, 120, 122, 123, 137, 138, 140, 143, 145, 149, 203–205
OACI	Organisation de l'Aviation Civile Internationale, 115, 121, 145
Plan de vol	Le plan de vol est un résumé du trajet prévu d'un aéronef avant son décollage, 114–116, 123, 126, 127, 137, 203, 205, 207
PROBE	Population Reinforced Optmization Based Exploration, 80
Route	Route aérienne, chemin pris par les avions et qui suit des balises de navigation, 113–116, 122, 125, 140, 150, 203, 204
Secteur	Secteur de contrôle, volume d'epace aérien limité sous la responsabilité d'un contrôleur aérien, 3, 116, 118–120, 122–127, 130, 132, 135, 137, 138, 140, 143, 145, 150, 186, 187, 203, 204, 206, 207
SHEM	Sorted Heavy Edge Matching, 48
UTC	United Time Coordinated, 126
VFR	Visual Flight Rules, 113
VLSI	Very-large-scale integration, 16, 38, 47
Zone de qualification	Ensemble de secteurs sur lequel un contrôleur aérien est habilité à contrôler, 3, 118–120, 122–127, 130, 135, 137, 138, 140, 143, 145, 149, 186

Liste des algorithmes

Références bibliographiques

[ALA99] S. Mounir ALAOUI, O. FRIEDER et T. EL-GHAZAWI : A Parallel Genetic Algorithm for task mapping on parallel machines. *In Proceedings of the IEEE IPDPS Workshop on Bio-Inspired Solutions to Parallel Processing Problems in conjunction*, 1999. *Cité p* 78

[ALL96] Jean-Marc ALLIOT : *Techniques d optimisation stochastique appliquées aux problèmes du controle aérien*. Thèse de doctorat, Thèse d'habilitation INPT, 1996. *Cité p* 123

[ALO85] N. ALON et V.D. MILMAN : λ_1, isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory, series B*, 38(1):73–88, 1985. *Cité p* 29

[ALP95a] Charles J. ALPERT et Andrew B. KAHNG : Recent directions in netlist partitioning : A survey. *Integration, the VLSI Journal*, 19(12):1–81, 1995. *Cité p* 4, 17

[ALP95b] Charles J. ALPERT et So-Zen YAO : Spectral partitioning : the more eigenvectors the better. *In Proceedings of the ACM/IEEE Design Automation Conference*, pages 195–200, 1995. *Cité p* 33

[ALP96a] C. J. ALPERT et A. B. KAHNG : A Hybrid Multilevel/Genetic Approach for Circuit Partitioning. *In Proceedings of the SIGDA Physical Design Workshop*, pages 100–105, 1996. *Cité p* 81

[ALP96b] Charles J. ALPERT, Lars W. HAGEN et Andrew B. KAHNG : A Hybrid Multilevel/Genetic Approach for Circuit Partitioning. *In Proceedings of the IEEE Asia Pacific Conference on Circuits and Systems*, pages 298–301, 1996. *Cité p* 44

[ALP96c] Charles J. ALPERT, Lars W. HAGEN et Andrew B. KAHNG : A Hybrid Multilevel/Genetic Approach for Circuit Partitioning. *In IEEE Asia Pacific Conference on Circuits and Systems*, pages 298–301, 1996. *Cité p* 76

[ALP97] Charles J. ALPERT, Jen-Hsin HUANG et Andrew B. KAHNG : Multilevel Circuit Partitioning. *In Proceedings of the ACM/IEEE Design Automation Conference*, pages 530–533, 1997. *Cité p* 50

[APE02] Adriana APETREI et Ovidiu GHEORGHIES : An Evolutionary Approach to Graph Partitioning, 2002. *Cité p* 78

[BAR93] Stephen T. BARNARD et Horst D. SIMON : A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *In Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computing*, pages 711–718, 1993. *Cité p* 15, 29, 44, 45

[BAR94] Stephen T. BARNARD et Horst D. SIMON : A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *Concurrency : Practice and Experience*, 6:101–107, 1994. *Cité p* 15, 29, 32, 33, 173

[BAR04] Mousbah BARAKE, Pierre CHARDAIRE et Geoff P. MCKEOWN : The probe metaheuristic and its application to the multiconstraint knapsack problem. *Applied Optimization*, 86:19–36, 2004. *Cité p* 80

[BAT99] R. BATTITI, A. BERTOSSI et A. CAPPELLETTI : Multilevel Reactive Tabu Search for Graph Partitioning. Rapport technique UTM 554, 1999. *Cité p* 81

[BAÑ03] R. BAÑOS, C. GIL, J. ORTEGA et F.G. MONTOYA : Multilevel Heuristic Algorithm for Graph Partitioning. *In Proceedings of the European Workshop on Evolutionary Computation in Combinatorial Optimization*, pages 143–153, 2003. *Cité p* 50, 62, 173

[BEN05] Florence BENEZIT, Timothee COUR et Jianbo SHI : Spectral Segmentation with Multi-Scale Graph Decomposition. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2005. *Cité p* 5, 17, 34, 151

[BER02] Pavel BERKHIN : Survey Of Clustering Data Mining Techniques. Rapport technique, Accrue Software, San Jose, CA, 2002. *Cité p* 5, 26

[BEZ99] S.L. BEZRUKOV, R. ELSÄSSER et U.-P. SCHROEDER : On Bounds for the k-Partitioning of Graphs. *In Computing and Combinatorics : 5th Annual International Conference, COCOON'99*, volume 1627 de *Lecture Notes in Computer Science*, page 154, 1999. *Cité p* 29

[BIC04a] Charles-Edmond BICHOT : Optimisation du découpage de l'espace aérien par diverses métaheuristiques. Mémoire de D.E.A., DEA Systèmes informatique du LAAS, 2004. *Cité p* 27, 118, 126

[BIC04b] Charles-Edmond BICHOT, Jean-Marc ALLIOT, Nicolas DURAND et Pascal BRISSET : Optimisation par fusion et fission. Application au problème du découpage aérien européen. *Journal Européen des Systèmes Automatisés (JESA)*, 38(9-10):1141–1173, 2004. *Cité p* 62, 90, 109

[BIC05] Charles-Edmond BICHOT et Jean-Marc ALLIOT : A theoretical approach to defining the European Core Area. Rapport technique, École Nationale de l'Aviation Civile / Centre d'Étude de la Navigation Aérienne, France, 2005. *Cité p* 27, 118, 143

[BIC06a] Charles-Edmond BICHOT : A metaheuristic based on fusion and fission for partitioning problems. *In Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium*, 2006. *Cité p* 90, 109, 129

[BIC06b] Charles-Edmond BICHOT : Metaheuristics versus spectral and multilevel methods applied on an Air Traffic Control problem. *In Proceedings of the 12th IFAC Symposium on Information Control Problems in Manufacturing (INCOM)*, pages 493–498, may 2006. *Cité p* 18

[BIC07] Charles-Edmond BICHOT : A new Method, the Fusion Fission, for the relaxed k-way graph partitioning problem, and comparisons with some Multilevel algorithms. *Journal of Mathematical Modeling and Algorithms (JMMA)*, 6(3):319–344, 2007. *Cité p* 18, 64, 90, 92, 130, 131

[BLA03] Daniel BLANC : *Précis de physique nucléaire*. Dunod, 2^{e} édition, 2003. *Cité p* 85

[BRI05] Michael BRINKMEIER : A Simple and Fast Min-cut Algorithm. *In Proceedings of the Fundamentals of Computation Theory*, pages 317–328, 2005. *Cité p* 26

[BRO57] S.R. BROADBENT et J.M. HAMMERSLEY : Percolation Processes I. Crystals and Mazes. *Proceedings of the Cambridge Philosophical Society*, 53(3):629–641, 1957. *Cité p* 51

[BUI89] Thang BUI, Christopher HEIGHAM, Curt JONES et Tom LEIGHTON : Improving the performance of the Kernighan-Lin and simulated annealing graph bisection algorithms. *In Proceedings of the 26th ACM/IEEE Design Automation Conference*, pages 775–778, 1989. *Cité p* 38

[BUI93] T. BUI et C. JONES : A heuristic for reducing fill in sparse matrix factorization. *In Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computing*, pages 445–452, 1993. *Cité p* 44, 47

[CER94] R. CERF : *Une théorie asymptotique des algorithmes génétiques*. Thèse de doctorat, université de Montpellier II, 1994. *Cité p* 75

[CFM00] *Basic CFMU Handbook - General & CFMU Systems*, Eurocontrol CFMU, 6.0 édition, février 2000. *Cité p* 207

[CHA07] P. CHARDAIRE, M. BARAKE et G. P. MCKEOWN : A PROBE based heuristic for Graph Partitioning. *IEEE Transaction on Computers*, 2007. in submission process. *Cité p* 19, 80, 173

[CHE06] Cédric CHEVALIER et François PELLEGRINI : Improvement of the Efficiency of Genetic Algorithms for Scalable Parallel Graph Partitioning in a Multi-Level Framework. *In Proceedings of Euro-Par06*, volume 4128 de *Lecture Notes in Computer Science*, pages 243–252, 2006. *Cité p* 76

[COL92] A. COLORNI, M. DORIGO et V. MANIEZZO : Distributed Optimization by Ant Colonies. *In Proceedings of the First European Conference on Artificial Life*, pages 134–142. Elsevier Publishing, 1992. *Cité p* 66

[COM06] Francesc COMELLAS et Emili SAPENA : A multiagent algorithm for graph partitioning. *In EvoWorkshops*, Lecture Notes in Computer Science, pages 279–285, 2006. *Cité p* 81

[CON03] J. CONG, M. ROMESIS et M. XIE : Optimality, Scalability and Stability Study of Partitioning and Placement Algorithms. *In Proceedings of the International Symposium on Physical Design*, pages 88–94, 2003. *Cité p* 4

[DAC04] Huy Tran DAC : *Sectorisation contrainte de l'espace aérien*. Thèse de doctorat, Université de Technologie de Compiègne, 2004. *Cité p* 120

[DEJ93] Kenneth A. DE JONG et Jayshree SARMA : Generation Gaps Revisited. *In* L. Darrell WHITLEY, éditeur : *Foundations of Genetic Algorithms 2*, pages 19–28. Morgan Kaufmann, 1993. *Cité p* 75

[DEL95] Daniel DELAHAYE : *Optimisation de la sectorisation de l'espace aérien par algorithmes génétiques*. Thèse de doctorat, École Nationale Supérieure de l'aéronautique et de l'espace, 1995. *Cité p* 3, 120, 123

[DGA07] DGAC, éditeur. *Réglementation de la circulation aérienne*. Service de l'Information Aéronautique, 2007. *Cité p* 115

[DHI01] Inderjit S. DHILLON : Co-clustering documents and words using bipartite spectral graph partitioning. *In Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 269–274, 2001. *Cité p* 5, 17, 156, 157

[DHI04a] Inderjit S. DHILLON, Yuqiang GUAN et Brian KULLIS : Kernel k-means, Spectral Clustering, and Normalized Cuts. *In Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 551–556, 2004. *Cité p* 29, 193

[DHI04b] Inderjit S. DHILLON, Yuqiang GUAN et Brian KULLIS : A Unified View of Kernel k-means, Spectral Clustering and Graph Cuts. Rapport technique TR-04-25, University of Texas at Austin, 2004. *Cité p* 18

[DHI04c] Inderjit S. DHILLON, Yuqiang GUAN et Brian KULLIS : A Unified View of Kernel k-means, Spectral Clustering and Graph Partitioning. Rapport technique TR-04-25, University of Texas at Austin, 2004. *Cité p* 26, 29, 193

[DHI05] Inderjit S. DHILLON, Yuqiang GUAN et Brian KULIS : A fast kernel-based multilevel algorithm for graph clustering. *In Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 629–634, 2005. *Cité p* 5

[DHI07] Inderjit S. DHILLON, Yuqiang GUAN et Brian KULIS : Weighted Graph Cuts without Eigenvectors : A Multilevel Approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007. To appear. *Cité p* 5, 18, 26, 188, 193, 194

[DIE95] R. DIEKMANN, B. MONIEN et R. PREIS : Using Helpful Sets to Improve Graph Bisections. *In Proceedings of the DIMACS Workshop on Interconnection Networks and Mapping and Scheduling Parallel Computations*, pages 57–73, 1995. *Cité p* 50, 193

[DIE96] Ralf DIEKMANN, Reinhard LÜLING, Burkhard MONIEN et Carsten SPRÄNER : Combining Helpful Sets and Parallel Simulated Annealing for the Graph-Partitioning Problem. *Parallel Algorithms and Applications*, 8:61–84, 1996. *Cité p* 50, 62

[DIE98] Ralf DIEKMANN, Robert PREIS, Frank SCHLIMBACH et Chris WALSHAW : Aspect Radio for Mesh Partitioning. *In Euro-Par*, volume 1470 de *Lecture Notes in Computer Science*, 1998. *Cité p* 51

[DIN01a] Chris DING, Xiaofeng HE, Hongyuan ZHA, Ming GU et Horst D. SIMON : A Min-max Cut Algorithm for Graph Partitioning and Data Clustering. *In Proceedings of the IEEE International Conference on Data Mining*, pages 107–114, 2001. *Cité p* 5, 17

[DIN01b] Chris DING, Xiaofeng HE, Hongyuan ZHA, Ming GU et Horst D. SIMON : A Min-max Cut Algorithm for Graph Partitioning and Data Clustering. *In Proceedings of the IEEE International Conference on Data Mining*, pages 107–114, 2001. *Cité p* 29, 34

[DON72] W.E. DONATH et A.J. HOFFMAN : Algorithms for partitioning graphs and computer logic based on eigenvectors of connection matrices. *IBM Technical Disclosure Bulletin*, 15(3):938–944, 1972. *Cité p* 29

[DON73] W.E. DONATH et A.J. HOFFMAN : Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, pages 420–425, 1973. *Cité p* 29, 32

[DOR99a] M. DORIGO, G. Di CARO et L.M. GAMBARDELLA : Ant Algorithms for Discrete Optimization. *Artificial Life*, 5(2):137–172, 1999. *Cité p* 66

[DOR99b] Marco DORIGO et Gianni DI CARO : The Ant Colony Optimization Meta-Heuristic. *In* David CORNE, Marco DORIGO et Fred GLOVER, éditeurs : *New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999. *Cité p* 66

[DOR02] M. DORIGO et T. STÜTZLE : The ant colony optimization metaheuristic : Algorithms, Applications, and Advances. *In* F. GLOVER et G. KOCHENBERGER., éditeurs : *Handbook of Metaheuristics*. Kluwer, 2002. *Cité p* 66, 67

[DOR05] Marco DORIGO et Christian BLUM : Ant colony optimization theory : A survey. *Theoretical Computer Science*, 344(2-3):243–278, 2005. *Cité p* 66

[DRÉ03] Johann DRÉO, Alain PÉTROWSKI, Patrick SIARRY et Eric TAILLARD : *Métaheuristiques pour l'optimisation difficile*. Eyrolles, 2003. *Cité p* 59, 60, 67, 71, 74

[DUR04] Nicolas DURAND : *Algorithmes Génétiques et autres méthodes d'optimisation appliqués à la gestion de trafic aérien*. Thèse de doctorat, Thèse d'habilitation INPT, 2004. *Cité p* 76

[DUT93] S. DUTT : New Faster Kernighan-Lin-Type Graph-Partitioning Algorithms. *In Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 370–377, 1993. *Cité p* 38

[EIB99] Ágoston Endre EIBEN, Robert HINTERDING et Zbigniew MICHALEWICZ : Parameter Control in Evolutionnary Algorithms. *IEEE Transaction on Evolutionary Computation*, 3(2):124–141, 1999. *Cité p* 74

[ELS03] Robert ELSÄSSER, Thomas LÜCKING et Burkhard MONIEN : On Spectral Bounds for the k-Partitioning of Graphs. *Theory of Computing Systems*, 36(5):461–478, 2003. *Cité p* 29, 32

[EUR05] Eurocontrol final report on European Commission's mandate to support the establishment of functional airspace blocks, may 2005. *Cité p* 121

[EUR06a] EUROCONTROL : The impact of fragmentation in European ATM/CNS. Rapport technique, Eurocontrol, apr 2006. *Cité p* 125

[EUR06b] EUROCONTROL : Performance Review Report. Rapport technique, Eurocontrol, Performance Review Commission, apr 2006. *Cité p* 125

[FAB07] FAB Europe Central — Redéfinir le contrôle du trafic aérien au coeur de l'Europe, production conjointe Belgocontrol, DSNA, DFS, LAA, LVNL, MUAC et skyguide, 2007, 2007. *Cité p* 122

[FEL04] Pedro F. FELZENSZWALB et Daniel P. HUTTENLOCHER : Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004. *Cité p* 5, 17

[FID82] C. M. FIDUCCIA et R. M. MATTHEYSES : A Linear-Time Heuristic for Improving Network Partitions. *In Proceedings of 19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982. *Cité p* 38, 50, 51

[FIE75] Miroslav FIEDLER : A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25, 1975. *Cité p* 32

[FOG66] L. J. FOGEL, A. J. OWENS et M. J. WALSH : *Artificial Intelligence through Simulated Evolution*. Wiley, 1966. *Cité p* 71, 73

[FOR56] L. R. FORD et D. R. FULKERSON : Maximal flow through a network. *Canadian Journal of Mathematic*, 8:399–404, 1956. *Cité p* 26

[FOR03] David FORSYTH et Jean PONCE : *Computer Vision – A modern approach*, chapitre Segmentation using clustering methods. Prentice-Hall, 2003. *Cité p* 152

[FRA57] Alex FRASER : Simulation of genetic systems by automatic digital computers. *Australian Journal of Biological Sciences*, 10:484–491, 1957. *Cité p* 71

[GAR79] Michael R. GAREY et David S. JOHNSON : *Computers and intractability : A Guide to the Theory of NP-Completeness*. Freeman, 1979. *Cité p* 20, 21

[GDA01] Yoram GDALYAHU, Daphna WEINSHALL et Michael WERMAN : Self-Organization in Vision : Stochastic Clustering for Image Segmentation, Perceptual Grouping, and Image Database Organization. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 23(10):1053–1074, 2001. *Cité p* 17

[GIA04] David GIANAZZA : *Optimisation des flux de trafic aérien*. Thèse de doctorat, Institut National Polytechnique de Toulouse, 2004. *Cité p* 4

[GIL06] C. GIL, R. BAÑOS, M. G. MONTOYA et J. GOMEZ : Performance of Simulated Annealing, Tabu Search, and Evolutionary Algorithms for Multi-objective Network Partitioning. *Algorithmic Operations Research*, 1:55–64, 2006. *Cité p* 17, 62

[GOL89] David GOLDBERG : *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Welsey, 1989. *Cité p* 71, 75

[GOL96] G.H. GOLUB et C.F. Van LOAN : *Matrix Computations*. Johns Hopkins University Press, Baltimore, 3rd édition, 1996. *Cité p* 32

[GOT04] Jean-Baptiste GOTTELAND : *Optimisation du trafic au sol sur les grands aéroports*. Thèse de doctorat, Institut National Polytechnique de Toulouse, 2004. *Cité p* 76, 116

[GUA98] Stephen GUATTERY et Gary L. MILLER : On the Quality of Spectral Separators. *SIAM Journal on Matrix Analysis and Applications*, 19(3):701–719, 1998. *Cité p* 29

[GUP97] Anshul GUPTA : Fast and Effective Algorithms for Graph Partitioning and Sparse Matrix Ordering. *IBM Journal of Research and Development*, 41(1/2):171–184, 1997. *Cité p* 49

[HAG92a] Lars HAGEN et Andrew B. KAHNG : New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design*, 11(9):1074–1085, 1992. *Cité p* 29, 30, 32, 34

[HAG92b] Lars W. HAGEN et Andrew B. KAHNG : A new approach to effective circuit clustering. *In Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 422–427, 1992. *Cité p* 17, 47

[HAG97] Lars W. HAGEN, Dennis J.-H. HUANG et Andrew B. KAHNG : On implementation choices for iterative improvement partitioning algorithms. *IEEE Transaction on Computer Aided Design*, 16(10):1199–1205, 1997. *Cité p* 38

[HAL05] Anders HALLGREN : Restructuring European airspace : functional airspace blocks. *Skyway*, pages 20–22, autumn 2005. *Cité p* 123

[HEN93] Bruce HENDRICKSON et Robert LELAND : Multidimensional Spectral Load Balancing. Rapport technique SAND93-0074, Sandia National Laboratories, 1993. *Cité p* 29, 30, 33

[HEN95a] Bruce HENDRICKSON et Robert LELAND : *The Chaco User's Guide*. Sandia National Laboratories, 2.0 édition, 1995. *Cité p* 34, 191

[HEN95b] Bruce HENDRICKSON et Robert LELAND : An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations. *SIAM Journal on Scientific Computing*, 16(2):452–469, 1995. *Cité p* 29, 32, 33, 50, 51, 191

[HEN95c] Bruce HENDRICKSON et Robert W. LELAND : A Multilevel Algorithm For Partitioning Graphs. *In Proceedings of Supercomputing*, 1995. *Cité p* 32, 38, 41, 44, 47, 192

[HEN97] Bruce HENDRICKSON, Robert LELAND et Rafael Van DRIESSCHE : Skewed graph partitioning. *In Proceedings of the 8th SIAM conference on parallel processing for scientific computing*, 1997. *Cité p* 192

[HEN98] Bruce HENDRICKSON : Graph Partitioning and Parallel Solvers : Has the Emperor No Clothes ? (Extended Abstract). *In Proceedings of the Workshop on Parallel Algorithms for Irregularly Structured Problems*, pages 218–225, 1998. *Cité p* 15

[HOL62] John HOLLAND : Outline for a Logical Theory of Adaptive Systems. *Journal of the ACM*, 9(3):297–314, 1962. *Cité p* 71

[HOL81] Ian HOLYER : The NP-Completness of Some Edge-Partition Problems. *SIAM Journal of Computing*, 10(4):713–717, 1981. *Cité p* 11

[HU98] Y. F. HU, R. J. BLAKE et D. R. EMERSON : An optimal migration algorithm for dynamic load balancing. *Concurrency-Practice and Experience*, 10(6):467–483, 1998. *Cité p* 27, 42, 43, 53

[HYA73] L. HYAFIL et R.L. RIVEST : Graph partitioning and constructing optimal decision trees are polynomial complete problems. Rapport technique 33, IRIA-Laboria, Rocquencourt. France, oct 1973. *Cité p* 20

[ING89] L. INGBER : Very fast simulated re-annealing. *Mathematical Computer Modelling*, 12(8): 967–973, 1989. *Cité p* 61

[JAI99] A. K. JAIN, M. N. MURTY et P. J. FLYNN : Data clustering : a review. *ACM Computing Surveys*, 31(3):264–323, 1999. *Cité p* 26, 157

[JOH89] David S. JOHNSON, Cecilia R. ARAGON, Lyle A. MCGEOCH et Catherine SCHEVON : Optimization by simulated annealing : an experimental evaluation ; Part I, graph partitioning. *Operations Research Society of America*, 37(6):865–892, 1989. *Cité p* 62, 64, 65

[KAN02] T. KANUNGO, D. MOUNT, N. NETANYAHU, C. PIATKO, R. SILVERMAN et A. WU : An efficient k-means clustering algorithm : analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002. *Cité p* 26

[KAR95] George KARYPIS et Vipin KUMAR : Analysis of Multilevel Graph Partitioning. *In Proceedings of Supercomputing*, 1995. *Cité p* 44, 48, 50

[KAR98a] George KARYPIS et Vipin KUMAR : A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal of Scientific Computing*, 20(1):359–392, 1998. *Cité p* 28, 29, 37, 38, 45, 47, 48, 50, 133, 192

[KAR98b] George KARYPIS et Vipin KUMAR : *Metis : A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*. University of Minnesota, 4.0 édition, sep 1998. *Cité p* 48, 79, 192

[KAR98c] George KARYPIS et Vipin KUMAR : Multilevel Algorithms for Multi-Constraint Graph Partitioning. *In Proceedings of Supercomputing*, pages 1–13, 1998. *Cité p* 38

[KAR98d] George KARYPIS et Vipin KUMAR : Multilevel k-way Partitioning Scheme for Irregular Graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998. *Cité p* 15, 38, 41, 42, 50, 51, 192

[KER70] B. W. KERNIGHAN et S. LIN : An Efficient Heuristic Procedure for Partitioning Graphs. *Bell System Technical Journal*, 49(2):291–307, 1970. *Cité p* 4, 14, 15, 17, 25, 28, 35, 38, 50

[KIR83] S. KIRKPATRICK, C. D. GELATT et M. P. VECCHI : Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983. *Cité p* 60

[KOR04] Peter KOROšEC, Jurij ŠILC et Borut ROBIč : Solving the mesh-partitioning problem with an ant-colony algorithm. *Parallel Computing*, 30(5-6):785–801, 2004. *Cité p* 81, 101

[KRI84] Balakrishnan KRISHNAMURTHY : An Improved Min-Cut Algorithm for Partitioning VLSI Networks. *IEEE Transactions on Computers*, 33(5):438–446, 1984. *Cité p* 38

[LAN99] A. E. LANGHAM et P. W. GRANT : A Multilevel k-way Partitioning Algorithm for Finite Element Meshes using Competing Ant Colonies. *In Proceedings of the ACM Genetic and Evolutionary Computation Conference*, volume 2, pages 1602–1608, 1999. *Cité p* 81

[LAN04a] Kevin LANG : Finding good nearly balanced cuts in power law graphs. Rapport technique YRL-2004-036, Yahoo ! Research Labs, november 2004. *Cité p* 80

[LAN04b] Kevin LANG et Satish RAO : A Flow-Based Method for Improving the Expansion or Conductance of Graph Cuts. *In Proceedings of Integer Programming and Combinatorial Optimization*, volume 3064 de *Lecture Notes in Computer Science*, pages 325–337, 2004. *Cité p* 173

[LOR02] Helena R. LORENCO, Olivier MARTIN et Thomas STÜTZLE : Iterated Local Search. *In* F. GLOVER et G. KOCHENBERGER., éditeurs : *Handbook of Metaheuristics*. Kluwer, 2002. *Cité p* 80

[MAI94] Harpal MAINI, Kishan MEHROTRA, Chilukuri K. MOHAN et Sanjay RANKA : Genetic algorithms for graph partitioning and incremental graph partitioning. *In Proceedings of Supercomputing*, pages 449–457, 1994. *Cité p* 76, 78

[MAN96] T.W. MANIKAS et J.T. CAIN : Genetic Algorithms vs. Simulated Annealing : A Comparison of Approaches for Solving the Circuit Partitioning Problem. Rapport technique TR-96-101, University of Pittsburgh, 1996. *Cité p* 78

[MAR96] Oliver C. MARTIN et Steve W. OTTO : Combining Simulated Annealing with Local Search Heuristics. *Annals of Operations Research*, 63:57–75, 1996. *Cité p* 50, 62

[MAR01] David R. MARTIN, Charless FOWLKES, Doron TAL et Jitendra MALIK : A Database of Human Segmented Natural Images and its Application. *In Proceedings of the IEEE International Conference on Computer Vision*, pages 416–425, 2001. *Cité p* 152

[MAR04] Aleix M. MARTÍNEZ, Pradit MITTRAPIYANURUK et Avinash C. KAK : On Combining Graph-Partitioning with Non-Parametric Clustering for Image Segmentation. *Computer Vision and Image Understanding*, pages 72–85, 2004. *Cité p* 5, 17, 151

[MAR05] Jacob G. MARTIN : Subproblem Optimization by Gene Correlation with Singular Value Decomposition. *In Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 1507–1514, 2005. *Cité p* 19, 81, 173

[MAR06] Jacob G. MARTIN : Spectral techniques for graph bisection in genetic algorithms. *In Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation*, pages 1249–1256, 2006. *Cité p* 19, 81

[MEI00] Marina MEILA et Jianbo SHI : Learning Segmentation by Random Walks. *In Proceedings of the Neural Information Processing Systems*, pages 873–879, 2000. *Cité p* 17

[MET53] Nicholas METROPOLIS, Arianna W. ROSENBLUTH, Marshall N. ROSENBLUTH, Augusta H. TELLER et Edward TELLER : Equations of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953. *Cité p* 60

[MEY05] Henning MEYERHENKE et Stefan SCHAMBERGER : Balancing Parallel Adaptive FEM Computations by Systems of Linear Equations. *In Euro-Par*, volume 3648 de *Lecture Notes in Computer Science*, 2005. *Cité p* 27, 51

[MEY06] Henning MEYERHENKE et Stefan SCHAMBERGER : A Parallel Shape Optimizing Load Balancer. *In Euro-Par*, volume 4128 de *Lecture Notes in Computer Science*, 2006. *Cité p* 51

[MOH91] Bojan MOHAR : The Laplacian Spectrum of graphs. *In* E. ALAVI, G. CHARTRAND, O. OELLERMANN et A. SCHWENK, éditeurs : *Graph Theory, Combinatorics, and Applications*. Wiley, 1991. *Cité p* 30

[MOH97] Bojan MOHAR : Some applications of Laplace eigenvalues of graphs. *In Graph Symmetry : Algebraic Methods and Applications*, volume 497 de *NATO ASI Series C*, 1997. *Cité p* 11, 30

[MON04] Burkhard MONIEN et Stefan SCHAMBERGER : Graph Partitioning with the Party Library : Helpful-Sets in Practice. *In Proceedings of the Symposium on Computer Architecture and High Performance Computing*, pages 198–205, 2004. *Cité p* 50, 193

[NAG94] H. NAGAMOCHI, K. NICHIMURA et T. IBARAKI : Computing All Small Cuts in Undirected Networks. *In Proceedings of the 5th SIGAL International Symposium on Algorithms and Computation*, volume 834 de *Lecture Notes In Computer Science*, pages 190–198, 1994. *Cité p* 26

[PAJ01] Stéphane PAJOT : *Percolation et économie*. Thèse de doctorat, Université de Nantes, oct 2001. *Cité p* 51

[PAP07] D.A. PAPA et I.L. MARKOV : Hypergraph Partitioning and Clustering. *In* T. GONZALEZ, éditeur : *Approximation Algorithms and Metaheuristics*. CRC Press, 2007. *Cité p* 38

[PEL96] François PELLEGRINI et Jean ROMAN : Experimental Analysis of the Dual Recursive Bipartitioning Algorithm for Static Mapping. Rapport technique 1038-96, LaBRI, Université Bordeaux I, 1996. *Cité p* 38, 192

[PEL06] François PELLEGRINI : *Scotch and libScotch*. ENSEIRB - LaBRI, Université de Bordeaux I, 4.0 édition, jan 2006. *Cité p* 134, 192

[PEL07] François PELLEGRINI : A parallelisable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries. *In EuroPar*, volume 4641 de *Lecture Notes in Computer Science*, 2007. *Cité p* 27, 50, 51, 53

[PER98] P. PERONA et W. FREEMAN : A Factorization Approach to Grouping. *In Proceedings of the European Conference on Computer Vision*, volume 1406 de *Lecture Notes in Computer Science*, 1998. *Cité p* 17

[POR80] M.F. PORTER : An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980. *Cité p* 158

[POT90] Alex POTHEN, Horst D. SIMON et Kang-Pu LIOU : Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal on Matrix Analysis and Applications*, 11(3):430–452, 1990. *Cité p* 15, 29, 30, 32, 34, 191

[PRE98] Robert PREIS et Ralf DIEKMANN : *The Party Partitioning Library, User Guide*. University of Paderborn, 1.99 édition, oct 1998. *Cité p* 193

[RAN01] Jarmo RANTAKOKKO : A Local Refinement Algorithm for Data Partitioning. *In Proceedings of the 5th International Workshop on Applied Parallel Computing (2000)*, volume 1947 de *Lecture Notes in Computer Science*, pages 140–148, 2001. *Cité p* 38

[RIE94] B.M. RIESS, K. DOLL et F.M. JOHANNES : Partitioning Very Large Circuits Using Analytical Placement Techniques. *In Proceedings of the IEEE/ACM Design Automation Conference*, pages 646–651, 1994. *Cité p* 17

[ROB04] Borut ROBIČ, Peter KOROŠEC et Jurij ŠILC : *Ant Colony Optimization*, chapitre Ant Colonies and the Mesh-Partitioning Problem. MIT Ptress, 2004. *Cité p* 81

[ROD05] D. RODNEY, Alan J. SOPER et Chris WALSHAW : Multilevel Refinement for the Vehicle Routing Problem. *In Proceedings of PlanSIG 2005, 24th Annual Workshop of UK Planning & Scheduling Special Interest Group*, pages 96–97, 2005. *Cité p* 44

[RON05] Dorit RON, Sharon WISHKO-STERN et Achi BRANDT : An Algebraic Multigrid based Algorithm for Bisectioning General Graphs. Rapport technique MCS05-01, Weizmann Institute of Science, 2005. *Cité p* 19, 50, 173

[RUL03] Benoit RULLEAU : Simulation arithmétique Europe/USA et optimisation de la structure des zones de qualification. Mémoire de D.E.A., DEA Informatique EDSYS/LAAS, 2003. *Cité p* 77

[RUM02] Andreas RUMMLER et Adriana APETREI : Graph Partitioning Revised - A Multiobjective Perspective. *In Proceedings of the World Conference on Systemics, Cybernetics and Informatics*, 2002. *Cité p* 17

[RÈG04a] Règlement n° 549/2004 du Parlement Européen et du Conseil du 10 mars 2004 fixant le cadre pour la réalisation du ciel unique européen, Journal officiel de l'Union européene, 2004. *Cité p* 120

[RÈG04b] Règlement n° 550/2004 du Parlement Européen et du Conseil du 10 mars 2004 relatif à la fourniture de services de navigation aérienne dans le ciel unique européen, Journal officiel de l'Union européene, 2004. *Cité p* 120

[RÈG04c] Règlement n° 551/2004 du Parlement Européen et du Conseil du 10 mars 2004 relatif à l'organisation et à l'utilisation de l'espace aérien dans le ciel unique européen, Journal officiel de l'Union européene, 2004. *Cité p* 118, 120, 121, 122, 125

[RÈG04d] Règlement n° 552/2004 du Parlement Européen et du Conseil du 10 mars 2004 concernant l'interopérabilité du réseau européen de gestion du trafic aérien, Journal officiel de l'Union européene, 2004. *Cité p* 120

[SAN89] Laura A. SANCHIS : Multiple-Way Network Partitioning. *IEEE Transactions on Computers*, 38(1):62–81, 1989. *Cité p* 38, 41

[SAN93] Laura A. SANCHIS : Multiple-Way Network Partitioning with Different Cost Functions. *IEEE Transactions on Computers*, 42(12):1500–1504, 1993. *Cité p* 41

[SCH97] Kirk SCHLOEGEL, George KARYPIS et Vipin KUMAR : Multilevel Diffusion Schemes for Repartitioning of Adaptive Meshes. *Journal of Parallel and Distributed Computing*, 47(2):109–124, 1997. *Cité p* 53

[SCH99] Kirk SCHLOEGEL, George KARYPIS et Vipin KUMAR : A New Algorithm for Multi-objective Graph Partitioning. *In Proceedings of the European Conference on Parallel Processing*, pages 322–331, 1999. *Cité p* 17

[SCH01] K. SCHLOEGEL, G. KARYPIS et V. KUMAR : Wavefront Diffusion and LMSR : Algorithms for Dynamic Repartitioning of Adaptive Meshes. *IEEE Transactions on Parallel and Distributed Systems*, 12(5):451–466, 2001. *Cité p* 53

[SCH03] S. SCHAMBERGER : Improvements to the Helpful-Set Heuristic and a New Evaluation Scheme for Graphs-Partitioners. *In Proceedings of the International Conference on Computational Science and its Applications*, pages 49–59, 2003. *Cité p* 50, 193

[SEL03] Navaratnasothie SELVAKKUMARAN et George KARYPIS : Multi-Objective hypergraph-partitioning algorithms for cut and maximum subdomain-degree minimization. *In Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 726–733, 2003. *Cité p* 17

[SEL06] Navaratnasothie SELVAKKUMARAN et George KARYPIS : Multi-Objective hypergraph-partitioning algorithms for cut and maximum subdomain-degree minimization. *IEEE Transactions on Computer-Aided Design*, 25(3):504–517, 2006. *Cité p* 17

[SHE03] N. SHENTAL, A. ZOMET, T. HERTZ et Y. WEISS : Learning and inferring image segmentations using the gbp typical cut. *In Proceedings of the IEEE International Conference on Computer Vision*, 2003. *Cité p* 151, 154

[SHI98a] J. SHI, S. BELONGIE, T. LEUNG et J. MALIK : Image and Video Segmentation : The Normalized Cut Framework. *In Proceedings of the IEEE International Conference on Image Processing*, pages 943–947, 1998. *Cité p* 17, 34

[SHI98b] Jianbo SHI et Jitendra MALIK : Motion Segmentation and Tracking Using Normalized Cuts. *In Proceedings of the IEEE International Conference on Computer Vision*, pages 1154–1160, 1998. *Cité p* 17, 34, 151

[SHI00] Jianbo SHI et Jitendra MALIK : Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. *Cité p* 5, 14, 17, 21, 34, 151, 193

[SIA89] P. SIARRY et G. DREYFUS : *La méthode du recuit simulé : théorie et applications.* ESPCI - IDSET, 10 rue Vauquelin Paris, 1989. *Cité p* 60

[SIM91] Horst D. SIMON : Partitioning of Unstructured Problems for Parallel Processing. *Computing Systems in Engineering*, 2:135–148, 1991. *Cité p* 32, 33, 192

[SIM97] Horst D. SIMON et Shang-Hua TENG : How Good is Recursive Bisection ? *SIAM Journal on Scientific Computing*, 18(5):1436–1445, 1997. *Cité p* 16, 19, 20

[SKA94] Wladyslaw SKARBEK et Andreas KOSCHAN : Colour Image Segmentation — A Survey. Rapport technique, Institute for Technical Informatics, Technical University of Berlin, oct 1994. *Cité p* 5

[SON94] Jianjian SONG : A Partially Asynchronous and Iterative Algorithm for Dstributed Load Balancing. *Parallel Computing*, 20(6):853–868, 1994. *Cité p* 43

[SOP00] Alan J. SOPER, Chris WALSHAW et M. CROSS : A combined evolutionary search and multilevel optimisation approach to graph partitioning. Rapport technique 00/IM/58, University of Greenwich, 2000. *Cité p* 81

[SOP04a] Alan J. SOPER, Chris WALSHAW et M. CROSS : A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph Partitioning. *Journal of Global Optimization*, 29:225–241, 2004. *Cité p* 44

[SOP04b] Alan J. SOPER, Chris WALSHAW et M. CROSS : A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph-Partitioning. *Journal of Global Optimization*, 29:225–241, 2004. *Cité p* 81, 101

[STO97] Mechthlid STOER et Frank WAGNER : A Simple Min-Cut Algorithm. *Journal of the ACM*, 44(4):585–591, 1997. *Cité p* 26

[TAL91] E. G. TALBI et P. BESSIERE : A Parallel Genetic Algorithm for the Graph Partitioning Problem. *In Proceedings of the ACM International Conference on Supercomputing*, pages 312–320, 1991. *Cité p* 76, 78

[TOU99] Michel TOULOUSE, Krishnaiyan THULASIRAMAN et Fred GLOVER : Multi-level Cooperative Search : A New Paradigm for Combinatorial Optimization and an Application to Graph Partitioning. *In European Conference on Parallel Processing*, pages 533–542, 1999. *Cité p* 80

[TUA76] P. L. TUAN, H. S. PROCTER et G. J. COULURIS : Advanced productivity analysis methods for air traffic control operation. Rapport technique CA 64025, Standford Research Institute, dec 1976. *Cité p* 123

[WAL95] Chris WALSHAW, M. CROSS et M. G. EVERETT : A Localised Algorithm for Optimising Unstructured Mesh Partitions. *Intl. J. Supercomputer Appl.*, 9(4):280–295, 1995. *Cité p* 44

[WAL97] Chris WALSHAW, M. CROSS et M. G. EVERETT : Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes. *Journal of Parallel and Distributed Computing*, 47(2):102–108, 1997. *Cité p* 38

[WAL00a] Chris WALSHAW et M. CROSS : Mesh Partitioning : A Multilevel Balancing and Refinement Algorithm. *SIAM Journal on Scientific Computing*, 22:63–80, 2000. *Cité p* 38, 42, 49, 50, 51, 193

[WAL00b] Chris WALSHAW, M. CROSS et K. MCMANUS : Multiphase mesh partitioning. *Applied Mathematical Modeling*, 25:123–140, 2000. *Cité p* 49

[WAL01a] Chris WALSHAW : A Multilevel Approach to the Graph Colouring Problem. Tech. Rep. 01/IM/69, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, may 2001. *Cité p* 44, 60

[WAL01b] Chris WALSHAW et M. CROSS : A Multilevel Lin-Kernighan-Helsgaun Algorithm for the Travelling Salesman Problem. *Future Generation Computer Systems*, 17(5):601–623, 2001. *Cité p* 44, 60

[WAL02a] Chris WALSHAW : *The Jostle executable user guide*. University of Greenwich, 3.1 édition, jul 2002. *Cité p* 193

[WAL02b] Chris WALSHAW : A Multilevel Approach to the Travelling Salesman Problem. *Operations research*, 50(5):862–877, 2002. *Cité p* 44, 60

[WAL03] Chris WALSHAW : A Multilevel Algorithm for Force-Directed Graph Drawing. *Journal of Graph Algorithms and Applications*, 7(3):253–285, 2003. *Cité p* 44

[WAL04] Chris WALSHAW : Multilevel Refinement for Combinatorial Optimisation Problems. *Annals of Operations Research*, 131:325–372, 2004. *Cité p* 44, 60, 80, 82, 110, 173, 182

[WAN01] Song WANG et Jeffrey Mark SISKIND : Image Segmentation with Minimum Mean Cut. *In Proceedings of the IEEE International Conference on Computer Vision*, pages 517–524, 2001. *Cité p* 17

[WAN03] S. WANG et J. SISKIND : Image segmentation with ratio cut. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(6):675–690, 2003. *Cité p* 5, 14

[WEI89] Yen-Chuen WEI et Chung-Kuan CHENG : Towards efficient hierarchical designs by ratio cut partitioning. *In Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 298–301, 1989. *Cité p* 4, 14

[WEI99] Yair WEISS : Segmentation using Eigenvectors : A Unifying View. *In Proceedings of the IEEE International Conference on Computer Vision*, pages 975–982, 1999. *Cité p* 5, 34, 151

[WIL91] Roy D. WILLIAMS : Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations. *Concurrency*, 3:457–481, 1991. *Cité p* 62, 192

[YAN02] S. YANG : Adaptive Non-Uniform Crossover Based on Statistics for Genetic Algorithms. *In Proceedings of the Genetic and Evolutionnary Computation Conference*, pages 650–657, 2002. *Cité p* 74

[YAR00] Elie YARACK : An Evaluation of Move-Based Multi-Way Partitioning Algorithms. *In Proceedings of the 2000 IEEE International Conference on Computer Design : VLSI in Computers & Processors*, page 363, 2000. *Cité p* 15

[YU03] S. YU et J. SHI : Multiclass spectral clustering, 2003. *Cité p* 155

[ZHA01] Hongyuan ZHA, Xiaofeng HE, Chris H. Q. DING, Ming GU et Horst D. SIMON : Bipartite Graph Partitioning and Data Clustering. *In ACM Conference on Information and Knowledge Management*, pages 25–32, 2001. *Cité p* 5, 17

[ZHA02] Zhizi ZHAO, Lixin TAO et Yongchang ZHAO : An effective algorithm for multiway hypergraph partitioning. *IEEE Transactions on Circuits and Systems I : Fundamental Theory and Applications*, 49(8):179–1092, 2002. *Cité p* 38

[ZHA04] Ying ZHAO et George KARYPIS : Empirical and Theoretical Comparisons of Selected Criterion Functions for Document Clustering. *Machine Learning*, 55(3):311–331, 2004. *Cité p* 17