Optimisation Discrete

Solving the Max2Sat Problem

Sonia Cafieri

ENAC - École Nationale de l'Aviation Civile

sonia.cafieri@enac.fr

Outline

Boolean variables

Definition

A *boolean variable* x is a variable that can assume only two values 0 and 1.

0 is for *false*, 1 for *true*.

The *negation* $\bar{x}$ *of a boolean variable* x is the variable that assumes the value $1 - x$.

The *sum of boolean variables* $x_1 + x_2 + \cdots + x_k$ is defined to be 1 if at least one of the x_i's is 1 and 0 otherwise.

Note that $\bar{\bar{x}} = x$.

Literals and Clauses

Definition

A *literal* l is a variable x or its negation $\bar{x}$.

Let X be a set of boolean variables. For every $x \in X$ there are 2 literals over x, namely x itself and $\bar{x}$.

Definition

A *clause* over a set of boolean variables is a disjunction of literals.

Example:
suppose we have 2 variables x_1, x_2
we create a clause by using the *or* operator: $x_1 \vee \bar{x}_2$.

Literals and Clauses

Definition

The *length of a clause* is the number of its literals.

Example:

$$x_1 \vee \bar{x}_2 \vee \bar{x}_3$$

is a clause of length 3

its literals are x_1, $\bar{x}_2$ and $\bar{x}_3$

if $x_1 = 0, x_2 = 0, x_3 = 1$, then its value is 1.

CNF formula

Definition

A *boolean formula in Conjunctive Normal Form* (CNF) Φ is a conjunction of clauses:

$$\Phi = \bigwedge_{i=1}^{m} C_i, \quad \text{where } C_i \text{ are clauses, } \forall i \in \{1, .., m\}.$$

Example:

suppose we have 3 variables: x_1, x_2, x_3

we can create some clauses by using the *or* operator:

$$(x_1 \vee \bar{x}_2), (\bar{x}_1 \vee x_2), (\bar{x}_2 \vee \bar{x}_3), (x_1 \vee x_3), (\bar{x}_1 \vee \bar{x}_2), (\bar{x}_1 \vee x_2)$$

then we can join the clauses by using the *and* operator:

$$(x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2)$$

Definition

The *size* of a CNF formula is the sum of the length of all its clauses.

Assigment of values

Definition

An assigment of values to the set X of variables of a boolean formula is called a *truth assignment*.

An assignment of truth values to the propositional variables:

- *satisfies* a literal x_i if x_i takes the value 1
- *satisfies* a literal $\bar{x}_i$ if x_i takes the value 0
- *satisfies* a clause if it satisfies at least one literal of the clause
- *satisfies* a CNF formula if it satisfies all the clauses of the formula.

An assigmnent for a CNF formula Φ is *complete* if all the variables occurring in Φ have been assigned, otherwise it is *partial*.

The SAT problem

Definition

Given a boolean formula in Conjunctive Normal Form, consisting of a conjunction of m clauses C_i, each of which with 2 literals, the *2-Satisfiability Problem* (**2-SAT**), is the problem of deciding if there is a truth assignment to the literals such that each clause is satisfied (i.e. its value is 1).

Depending on whether this is possible or not, we say that the formula is satisfiable or unsatisfiable.

Each instance of the 2-SAT problem is a 2CNF formula.

The MAX2SAT problem

Definition

decision version

Given a boolean formula in Conjunctive Normal Form, consisting of a conjunction of m clauses C_i, each of which with 2 literals, and given an integer k, the *Maximum 2-Satisfiability problem*, (**MAX2SAT**), is the problem of deciding if there is a truth assignment to the literals such that *satisfies at least k clauses*.

Definition

optimization version

Given a boolean formula in Conjunctive Normal Form, consisting of a conjunction of m clauses C_i, each of which with 2 literals, the *Maximum 2-Satisfiability problem*, (**MAX2SAT**), is the problem of finding a truth assignment to the literals that *maximizes the number of satisfied clauses*.

The MAXSAT problem

The natural generalization of MAX2SAT is the MAXSAT problem.

Definition

The *Maximum Satisfiability problem* (**MAXSAT**) asks for the maximum number of clauses which can be satisfied by any assignment.
The clauses have not a limit on the number of literals.

There are several extensions to MAXSAT:

- The weighted maximum satisfiability problem (Weighted MAXSAT) asks for the maximum weight which can be satisfied by any assignment, given a set of weighted clauses.
- The partial maximum satisfiability problem (PMAXSAT) asks for the maximum number of clauses which can be satisfied by any assignment of a given subset of clauses. The rest of the clauses must be satisfied.

ENAC

Applications

SAT and MAXSAT are central problems in Artificial Intelligence, Logic and Computational Complexity.
Many important real-world applications:

- scheduling
- electronic design automation
- computer architecture design
- pattern recognition
- inference in Bayesian networks

MAXSAT as an Integer Programming problem

x_i binary variable :

- an unnegated literal is replaced by x_i,
- a negated literal is replaced by $1 - x_i$.

A clause is satisfied if and only if the sum of its literals is 1 or more.

Example:
the clause $\bar{v}_1 \vee v_2 \vee v_5 \vee \bar{v}_7$
is satisfied if: $(1 - x_1) + x_2 + x_5 + (1 - x_7) \geq 1$.

We have to handle the maximization of satisfied clauses.
A clause is either satisfied or it isn't:

$$y_j = 1 \quad \text{if clause } \varphi_j \text{ is satisfied}$$
$$y_j = 0 \quad \text{if clause } \varphi_j \text{ is not satisfied.}$$

IP formulation

Let Φ be a CNF formula: $\Phi = \varphi_1 \wedge \cdots \wedge \varphi_m$.
For all $j \leq m$, let $Y_j = \{i \leq n \mid x_i \in \varphi_i\}$ and $\bar{Y}_j = \{i \leq n \mid \bar{x}_i \in \varphi_i\}$ and $x \in \{0,1\}^n$, $y \in \{0,1\}^m$ be binary decision variables.

Integer Programming formulation for MAXSAT:

$$\begin{aligned} \max \quad & \sum_{j \leq m} y_j \\ \text{s.t. } \forall j \leq m \quad & \sum_{i \in Y_j} x_i + \sum_{i \in \bar{Y}_j} (1 - x_i) \geq y_j \\ & x \in \{0,1\}^n \\ & y \in \{0,1\}^m \end{aligned}$$

Complexity

- SAT was the first known NP-complete problem [Cook, 1971].
  (NP standing for Nondeterministic Polynomial time – any given solution of the problem p can be verified in polynomial time and every other problem in NP can be reduced to p).
- MAX2SAT is shown to be NP-complete by reduction from 3SAT.
- The more general MAXSAT is also NP-complete.
- 2SAT is polynomially solvable.

MAXSAT problems are difficult problems: the time required to solve the problem increases very quickly as the size of the problem grows.

How can we solve the problem?

A very simple algorithm: exhaustive search.
Enumerate all possible 1 and 0 configurations, and verify each one.

- As the number of variables becomes large, enumerating all possible combinations becomes impossible.
- For n variables, this search can require 2^n tests.
- Ex.: $n = 5$, there are $2^5 = 32$ possible configurations:
  00000, 00001, 00010, 00100, . . . , 11111.

Solution methods

Exact methods

- variants of Davis-Putnam-Logemann-Loveland (DPLL) procedure
  (designed for SAT - deterministic tree-search with backtracking) [Davis et al., 1962]
- Branch and Bound (BB) algorithms [Borchers and Furman, 1999].

Approximate methods

- GSAT
  (greedy local search – starts with a randomly generated assignment, then repeatedly changes ('flips') the assignment of the variable that leads to the largest decrease in the total number of unsatisfied clauses).
- WalkSAT
  (starts with randomly generated assignment, then repeatdly picks a clause which is unsatisfied by the current assignment and flips a variable within that clause).
- . . .

Branch and Bound algorithm

SAT problem

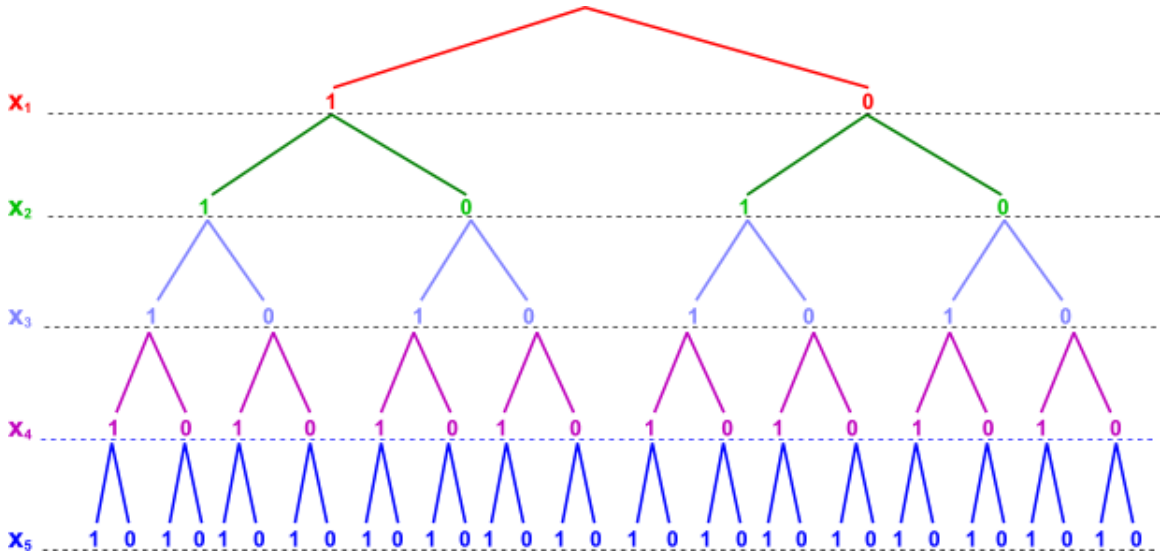
First consider the SAT problem.

Basic algorithm:

- Select a variable and set its value to 0 or 1.
  Note: different 0/1 assignments create branches of the search tree.
- Replace the selected variable with the selected value in the formula.
- If the selected value satisfies all the clauses, then assign a value to the next variable, and so on.
- Else, if the selected value fails to satisfy one of the clauses, *prune* the current branch and backtrack the search.
- Keep performing this process until all clauses are satisfied.

Search tree

Example of binary search tree



Branch and Bound algorithm

SAT problem

Example

$$(\bar{x}_2 \vee \bar{x}_5) \wedge (x_1 \vee \bar{x}_3) \wedge (x_4 \vee \bar{x}_5) \wedge (x_1 \vee x_2)$$

- Start setting $x_1 = 0$ and substitute this value into the formula.
- We get a 0 in two clauses, but we don't know yet if the formula is satisfied or not.
- Go to the next variable in the tree, x_2. Set $x_2 = 0$.
- The 4-th clause is unsatisfied, because has both x_1 and x_2 as 0.
  Therefore, we don't continue exploring that branch, since we cannot satisfy the formula.
- Try the other branch for x_2: set x_2 to 1. The 4-th clauses is now satisfied.
- Set $x_3 = 0$. The 2-nd clauses is satisfied.
- Set $x_4 = 0$.
- Set $x_5 = 0$. Then all clauses are satisfied: we solved the SAT problem.

Branch and Bound algorithm

SAT problem

Observation

Notice that we didn't need to go through every branch of the tree to find the solution.

Observation

If an assigned variable fails to satisfy one of the clauses, then we prune the corresponding branch of the search tree.

What does it happen when we consider MAXSAT?

Branch and Bound algorithm

MAXSAT problem

Consider now the MAXSAT problem.

- We want to satisfy the maximum number of clauses, not *all* the clauses.
- So, if we find an unsatisfied clause when assigning a value to a variable, then we have not to stop at that point and go back to the other branch.
- If a certain 0/1 value for a variable fails to satisfy a clauses, it could be that the same value satisfies many other clauses: it is possible that continuing down on that branch we get maximum satisfiability.
- We have to explore more branches!

When can we stop exploring a branch?

BB algorithm for MAX2SAT

Consider the IP formulation of the MAX2SAT problem.
Algorithm:

- At each node of the search tree solve the Linear Programming (LP) relaxation obtained by replacing the integrality requirements by the simple bounds:
  $\forall i \in \{1,\dots,n\}\ 0 \le x_i \le 1,\ \ \forall j \in \{1,\dots,m\}\ 0 \le y_i \le 1.$
- If the solution to the LP is infeasible, *backtrack* to a higher level in the search tree.
- If the solution to the LP is integral, compare it to the best integral solution found so far. *Backtrack* to a higher level in the search tree.
- Else (if the solution has one or more fractional variables) *branch* on one of the fractional variables and repeat the procedure.

BB algorithm for MAX2SAT

Some observation:

- *Branching* allows to extend the current partial assignment by instantiating a new variable. This leads to the creation of two new branches of the tree from the current branch: the left branch corresponds to instantiate the new variable to 0, the right branch corresponds to instantiate the new variable to 1.
- *Backtracking* is used to go back to a higher level in the search tree and continue exploring the tree along others branches.
- When we find an assignment of values such that the number of satisfied clauses is greater than the number we previously found, we store the values we have found. The corresponding value of the objective function is the *current bound* and it represents the *current best value* of the objective function.

BB algorithm for MAX2SAT

For the sake of simplicity, we only consider the basic BB algorithm for MAX2SAT, but usually suitable strategies are used to reduce the computational complexity.

Observation

BB algorithms for MAX2SAT usually implement methods to *reduce* the input formula: the formula is simplified according to *transformation rules* without changing the maximum number of satisfiable clauses
[Gramm, PhD thesis, 1999].

BB algorithm for MAX2SAT

Input a 2CNF instance

while (termination test not satisfied) **do**
  solve the continuous relaxation
  if (the solution is infeasible) **then**
    backtracking
  else
    if (the solution is integer) **then**
      if (the current value of the obj. func. is greater than the one previously found) **then**
        store the current solution
      endif
      backtracking
    else
      branching
    endif
  endif
endwhile

The output is the maximum number of satisfied clauses.

BB algorithm for MAX2SAT: problem relaxation

The linear continuous relaxation is obtained by relaxing the integrality constraints on the variables.

Let Φ be a CNF formula: $\Phi = \varphi_1 \wedge \cdots \wedge \varphi_m$.
For all $j \leq m$, let $Y_j = \{i \leq n \mid x_i \in \varphi_i\}$ and $\bar{Y}_j = \{i \leq n \mid \bar{x}_i \in \varphi_i\}$ and $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$ be decision variables.

$$\begin{aligned} \max \quad & \sum_{j \leq m} y_j \\ \text{s.t. } \forall j \leq m \quad & \sum_{i \in Y_j} x_i + \sum_{i \in \bar{Y}_j} (1 - x_i) \geq y_j \\ & 0 \leq x \leq 1 \\ & 0 \leq y \leq 1 \end{aligned}$$

BB algorithm for MAX2SAT: branching rule

We branch if we find a solution containing still fractional values.
A lot of branching rules are possible. What rule can we use?

A very simple branching rule:
branch on the first fractional variable x_i that we find, for $i \in \{1, \ldots, n\}$.
Using this criterion, branching index is:

$$i = \min \{1, \ldots, n\} \mid x_i \notin \{0, 1\}.$$

RandomWalk algorithm

An approximate method.

Basic algorithm:

- Randomly choose a truth assignment for the variables.
- If the current assignment satisfies the formula (for SAT) or the required number of clauses (for MAXSAT), then terminate the procedure.
- Else randomly pick a clause which is unsatisfied by the current assignment and flip a variable within such clause.

Note: GSAT, WalkSAT, ... , are variants of the RandomWalk algorithm.

A RandomWalk algorithm for MAX2SAT

Suppose we have MAX2SAT – decision version. We aim to satisfy k clauses.

Input a 2CNF instance and a positive integer k

randomly assign a 0/1 value to all variables
termination := 0
while (termination = 0) **do**
  replace the variables in the formula with the selected assignment
  if (at least k clauses are satisfied) **then**
    termination := 1
  else
    choose randomly an unsatisfied clause
    choose randomly a variable within the selected clause and change its value
  endif
  if (iteration limit reached) **then**
    termination := 1
  endif
endwhile

The output is the truth assignment that satisfies k clauses (if the solution is found).

A RandomWalk algorithm for MAX2SAT

Some observation.

- The algorithm does not guarantee finding an optimal solution.
- If a solution is found, it is usually quickly found.
- At each step, the clause to be changed can be randomly chosen.
- The fact that a clause is unsatisfied means that at least one of the variables in the clause must be flipped in order to reach a global solution.
- In some algorithms, the variable to be flipped is chosen from the selected unsatisfied clause by some greedy heuristic.
- A number of such heuristics have been studied. For ex., some algorithms take into account the number of currently satisfied clauses that would become unsatisfied if a variable were to be flipped.

Bibliography I

J. Gramm.
Exact algorithms for Max2Sat and their applications.
PhD thesis, 1999.

M. Davis, G. Logemann, D. Loveland.
A machine program for theorem-proving.
Communications of the ACM, 5:394–397, 1962.

T. Alsiniet, F. Manyá, J. Planes.
Improved Branch and Bound algorithms for Max-Sat.
Proceedings of the 6th International Conference on the Theory and Applications of Satisfiability Testing, 2003.

S. Joy, J.E. Mitchell, B. Borchers.
Solving MAX-SAT and Weighted MAX-SAT Problems Using Branch-and-Cut.
http://www.rpi.edu/ mitchj/papers.html, 1998.

ENAC