Optimisation Discrete

Solving the Knapsack Problem

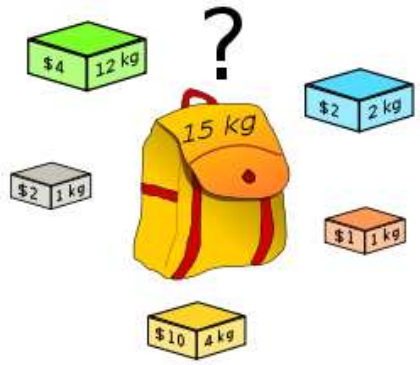
Sonia Cafieri

ENAC

sonia.cafieri@enac.fr

Outline

Knapsack problem (KP)



$$\begin{cases} \max & \sum_{i=1}^{n} x_i u_i \\ \text{s.t.} & \sum_{i=1}^{n} x_i v_i \leq V \\ & x_i \in \{0,1\} \quad \forall i \end{cases}$$

V = volume of the knapsack
v_i = volume of object i
u_i = value of object i
$x_i = 1$ if object i is selected, 0 otherwise

Solving the KP

There is no polynomial algorithm known that solves the problem to optimality.
There is also no proof that such an algorithm does not exist.

How to solve the Knapsack problem?

- Apply an algorithm with exponential running time to find an *optimal solution*
- Enumerate all possible solutions in a smart way to find an *optimal solution* : branch-and-bound or dynamical programming
- Apply a polynomial algorithm that constructs a *good but possibly non-optimal solution* : heuristic approach.

Heuristics for KP : Greedy algorithm

Greedy : algorithm that makes the choice that looks the best at the moment.

Idea :

Sort by most rewarding items with respect to a given criterion and take as many of the top most valuable items as will fit.

Heuristic 1 : sort u_i $i \in \{1, \ldots, n\}$ by non-increasing order

Heuristic 2 : sort v_i $i \in \{1, \ldots, n\}$ by increasing order

Heuristic 3 : sort u_i/v_i $i \in \{1, \ldots, n\}$ by non-increasing order

$\Rightarrow$ the most efficient greedy

Greedy algorithm for KP

Let i, j be indexes such that $j = 1, \ldots, n$ correspond to items sorted with respect to a given criterion

$W \leftarrow 0$
for $j = 1, \ldots, n$ **do**
  $W \leftarrow v_{i_j} + W$
  if $(W \leq V)$ **then**
    $x_{i_j} \leftarrow 1$
  else
    $x_{i_j} \leftarrow 0$
    $W \leftarrow W - v_{i_j}$
  endif
endfor

0-1 Knapsack and Continuous Knapsack

0-1 Knapsack problem :

you can only choose to take all or nothing of a particular item (0-1)

Continuous (Fractional) Knapsack :

you can take any fractions of the items to put in the knapsack

$\Rightarrow$ *continuous relaxation* of 0-1 Knapsack

For both versions :

Optimal Substructure property:

an optimal solution contains within it optimal solutions to subproblems.

If we remove item i from an optimal solution, then the remaining items must be optimal for the subproblem where the bag volume is $V - v_i$ and the available items do not include item i. Otherwise, we would be able to construct an even better solution than the optimal one.

Solving the CKP: Greedy (n.3)

Idea :

Sort by most rewarding items with respect to the ratio of value to size, and take as much as possible of the most valuable items as will fit.

Steps :

1. compute ratios (densities) u_i/v_i
2. sort them by non-increasing order
3. take as much as possible of the densest item not already in the bag

Complexity : $O(n \log n)$ for sorting, $O(n)$ for greedy choices

Solving the CKP: Greedy (n.3)

Sort values/volumes by non-increasing order.
Let i, j be indexes such that u_{i_j}/v_{i_j} with $j = 1$ is the maximum ratio:

$$\frac{u_{i_1}}{v_{i_1}} \geq \frac{u_{i_2}}{v_{i_2}} \geq \cdots \geq \frac{u_{i_n}}{v_{i_n}}$$

$W \leftarrow 0$
for $i \leq n$ **do**
  if $(W + v_{i_j} \leq V)$ **then**
    take item i :
      $x_{i_j} \leftarrow 1$
      $W \leftarrow W + v_{i_j}$
  else
    take a fraction of item i :
      $x_{i_h} \leftarrow \dfrac{V - W}{v_{i_h}}$
      $x_{i_j} \leftarrow 0 \qquad \forall j > h$
  endif
endfor

Solving the Continuous Knapsack Problem

Theorem

The Greedy algorithm (n.3) is optimal for the Continuous Knapsack Problem (CKP) (continuous relaxation of 0-1 Knapsack)

Proof. 1

Note that the algorithm fills the knapsack completely.
It may take a fraction of an item, which can only be the last selected item.

The greedy solution is such that there exists a h such that

$$1 = x_{i_1} = x_{i_2} = \cdots = x_{i_h} > x_{i_{h+1}} > x_{i_{h+2}} = \cdots = x_{i_n} = 0.$$

We compare this solution with another (feasible) solution $y_{i_1}, \ldots, y_{i_n}$.
Let k be the smallest index such that $y_{i_k} < 1$, and
let ℓ be the smallest index such that $k < \ell$ and $y_{i_\ell} > 0$.
(such an ℓ exists, otherwise the y solution would be equal to the x solution).

Solving the Continuous Knapsack Problem

Proof. (cont.)

We increase y_{i_k} and decrease y_{i_ℓ}, while keeping all other values equal.
Let $\epsilon = \min\{v_{i_k}(1 - y_{i_k}), v_{i_\ell} y_{i_\ell}\}$.

We increase y_{i_k} by $\frac{\epsilon}{v_{i_k}}$ and decrease y_{i_ℓ} by $\frac{\epsilon}{v_{i_\ell}}$, obtaining a new solution.

This new solution is no worse than the y solution.
Moreover, either y_{i_k} has become 1, or y_{i_ℓ} has become 0.

Repeating the same argument we find the solution x of the greedy.

Solving the Continuous Knapsack Problem

Theorem

The Greedy algorithm (n.3) is optimal for the Continuous Knapsack Problem (CKP) (continuous relaxation of 0-1 Knapsack)

Proof. 2

Let $O = \{o_1, o_2, \ldots, o_n\}$ be the optimal solution and $G = \{g_1, g_2, \ldots, g_n\}$ be the greedy solution (where the items are ordered with respect to the greedy choice).
We show that there exist an optimal solution such that it also takes the greedy choice in each iteration.

First, we show that an optimal solution selects (a fraction or 1 unit of) item g_1.

Solving the Continuous Knapsack Problem

Proof. 2 (cont.)

Two cases:

1. G takes 1 unit of g_1.
   If O also takes 1 unit of g_1, done.
   If O doesn't take 1 unit of g_1, then we take away volume v_{g_1} from O and put 1 unit of g_1 to it $\Rightarrow$ new solution O'.
   Of course O' has the same volume constraint V as O.
   Moreover, g_1 has maximum ratio u/v.
   $\Rightarrow$ O' is as good as O $\Rightarrow$ $g_1 \in O'$ and O' is an optimal solution.

2. G takes a fraction f of g_1 (the volume added to O is $f \times v_{g_1}$).
   If O also takes a fraction f of g_1, done.
   If O takes less than f of g_1 (cannot take more because o optimal), then we take away $f \times v_{g_1}$ from O and put f of g_1 to it $\Rightarrow$ new solution O'.
   As before, O' is as good as O (otherwise contradiction).

Solving the Continuous Knapsack Problem

Proof. 2 (cont.)

Second, we observe that
after each greedy choice is made, we have a problem of the same form as the original one.

The problem exhibits the optimal substructure property
$\Rightarrow$ $G \equiv O$.

Solving the Continuous Knapsack Problem

(Another) Alternative Proof:

The simplex algorithm for solving the CKP takes exactly the same steps as the greedy.

Let i_h be such that :

$$\begin{array}{lll} \forall j \in \{1,\ldots,h\} & x_{i_j} & \leftarrow \quad 1 \\ \forall j \in \{h+2,\ldots,n\} & x_{i_j} & \leftarrow \quad 0 \\ & x_{i_{h+1}} & \leftarrow \quad \dfrac{V - \sum_{j=1}^{h} v_{i_j}}{v_{i_{h+1}}} \end{array}$$

that is, i_h is such that:

$$\sum_{j=1}^{h} v_{i_j} \leq V \quad \text{and} \quad \sum_{j=1}^{h+1} v_{i_j} > V$$

Let us apply the simplex algorithm with the dictionary representation.

Solving the Continuous Knapsack Problem

(Another) Alternative Proof: (cont.)

$$\begin{cases} x_{n+1} &= 1 - x_{i_1} \\ x_{n+2} &= 1 - x_{i_2} \\ \vdots & \\ x_{2n} &= 1 - x_{i_n} \\ x_{2n+1} &= V - \sum_{j=1}^{n} v_{i_j} x_{i_j} \\ z &= \sum_{j=1}^{n} u_{i_j} x_{i_j} \end{cases}$$

Solving the Continuous Knapsack Problem

(Another) Alternative Proof: (cont.)

pivot: $x_{i_1} \leftrightarrow x_{n+1}$

$$\begin{cases} x_{i_1} & = & 1 - x_{n+1} \\ x_{n+2} & = & 1 - x_{i_2} \\ \vdots & & \\ x_{2n} & = & 1 - x_{i_n} \\ x_{2n+1} & = & V - \sum_{j=2}^{n} v_{i_j} x_{i_j} - v_{i_1}(1 - x_{n+1}) = (V - v_{i_1}) - \sum_{j=2}^{n} v_{i_j} x_{i_j} + v_{i_1} x_{n+1} \\ z & = & u_{i_1} + \sum_{j=2}^{n} u_{i_j} x_{i_j} - u_{i_1} x_{n+1} \end{cases}$$

We continue with following dictionaries.

Solving the Continuous Knapsack Problem

(Another) Alternative Proof: (cont.)

Dictionary h :
pivot: $xi_h \leftrightarrow x_{n+h}$

$$\begin{cases} x_{i_1} & = & 1 - x_{n+1} \\ x_{i_2} & = & 1 - x_{n+2} \\ \vdots & & \\ x_{i_h} & = & 1 - x_{n+h} \\ x_{n+h+1} & = & 1 - x_{i_{h+1}} \\ x_{2n} & = & 1 - x_{i_n} \\ x_{2n+1} & = & (V - \sum_{j=1}^{h} v_{i_j}) - \sum_{j=h+1}^{n} v_{i_j} x_{i_j} + \sum_{j=1}^{h} v_{i_j} x_{n+j} \\ z & = & \sum_{j=1}^{h} u_{i_j} + \sum_{j=h+1}^{n} u_{i_j} x_{i_j} - \sum_{j=1}^{h} u_{i_1} x_{n+j} \end{cases}$$

Solving the Continuous Knapsack Problem

(Another) Alternative Proof: (cont.)

Dictionary $h+1$:
pivot: $x_{i_{h+1}} \leftrightarrow x_{2n+1}$

$$\begin{cases} x_{i_1} = 1 - x_{n+1} \\ x_{i_2} = 1 - x_{n+2} \\ \vdots \\ x_{i_h} = 1 - x_{n+h} \\ x_{2n} = 1 - x_{i_n} \\ x_{i_{h+1}} = \dfrac{(V - \sum_{j=1}^{h} v_{i_j})}{v_{i_{h+1}}} - \dfrac{\sum_{j=h+2}^{n} v_{i_j} x_{i_j}}{v_{i_{h+1}}} + \dfrac{\sum_{j=1}^{h} v_{i_j} x_{n+j}}{v_{i_{h+1}}} - \dfrac{x_{2n+1}}{v_{i_{h+1}}} \\ x_{n+h+1} = 1 - (\text{expression of } x_{i_{h+1}}) \\ z = \sum_{j=1}^{h} u_{i_j} + u_{i_{h+1}}(\text{expression of } x_{i_{h+1}}) + \sum_{j=h+2}^{n} u_{i_j} x_{i_j} - \sum_{j=1}^{h} u_{i_j} x_{n+j} \end{cases}$$

Solving the Continuous Knapsack Problem

(Another) Alternative Proof: (cont.)

Dictionary $h+1$:
pivot: $x_{i_{h+1}} \leftrightarrow x_{2n+1}$

$$\begin{aligned} z = & \sum_{j=1}^{h} u_{i_j} + u_{i_{h+1}} \left(\frac{(V - \sum_{j=1}^{h} v_{i_j})}{v_{i_{h+1}}} \right) \\ & + \sum_{j=h+2}^{n} \left(u_{i_j} - u_{i_{h+1}} \frac{v_{i_j}}{v_{i_{h+1}}} \right) x_{i_j} \\ & + \sum_{j=1}^{h} \left(\frac{v_{i_j}}{v_{i_{h+1}}} - u_{i_j} \right) x_{n+j} \\ & - \frac{x_{2n+1}}{v_{i_{h+1}}} \end{aligned}$$

Solving the Continuous Knapsack Problem

(Another) Alternative Proof: (cont.)

We check the signs of the coefficients of the variables :

$$\forall j = h+2, \ldots, n \qquad \text{sign}\left(u_{i_j} - u_{i_{h+1}} \frac{v_{i_j}}{v_{i_{h+1}}}\right) = \text{sign}\left(\frac{u_{i_j}}{v_{i_j}} - \frac{u_{i_{h+1}}}{v_{i_{h+1}}}\right) = -1$$

because $j > h+1$

$$\forall j = 1, \ldots, h \qquad \text{sign}\left(\frac{u_{i_{h+1}}}{v_{i_{h+1}}} v_{i_j} - u_{i_j}\right) = \text{sign}\left(\frac{u_{i_{h+1}}}{v_{i_{h+1}}} - \frac{u_{i_j}}{v_{i_j}}\right) = -1$$

because $j < h+1$

Hence the simplex algorithm stops.

$\Rightarrow$ the (feasible) solution found is also the optimal solution.

Solving the KP exactly

Can we solve the KP exactly ?

$\Downarrow$

Branch-and-Bound (B&B)
(séparation et évaluation)

Branch and Bound

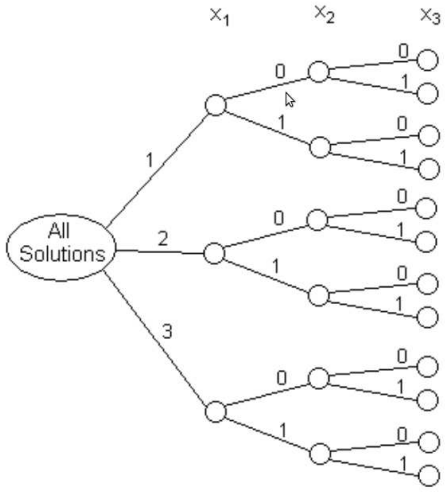
Branch-and-Bound (B&B) is a general framework for solving *exactly* integer and discrete optimization problems.

- *Intelligent* implicit enumeration of all possible solutions
- only potentially good solutions are enumerated,
  solutions that cannnot lead to improve the current solution are not explored
- based on the observation that the enumeration of integer solutions has a tree structure.

Branch and Bound - search tree

Search Tree :

- Nodes : represent stages of construction of the solution
- Arcs : represent choices made to build the solution



For the KP :

- Nodes : represent a step in which some objects have been placed in the bag, some others have been left out of the bag, and others for which no decision has yet been taken.
- Arcs : indicate the action of putting an object in the bag or not putting it in the bag

Branch and Bound - idea

The nodes of the search tree represent subsets of all of the possible solutions.
In particular, the root node represents *all* solutions that can be generated by growing the tree.

The main idea is to build step by step the search tree avoiding growing the whole tree as much as possible.

Only the most promising nodes are generated and explored at any stage.

The node that is the most promising is determined by estimating a bound on the best value of the objective function that can be obtained by growing that node to later stages.

Branch and Bound - main steps

- Branching :
  a node is selected for further growth and
  two new subproblems are created
- Bounding :
  upper and lower bounds on the best value
  attained by growing a node are estimated
- Pruning :
  nodes that are showed to be never either
  feasible or optimal are permanently discarded

Branch and Bound - main steps

During the search, a node must be in one of the following states :

(i) has no feasible solution

(ii) is such that its associated solution x_S is feasible and
  (a) x_S has a value better than the incumbent *
  (b) x_S has a value no better than the incumbent *

(iii) neither of states (i) and (ii) has yet been established.

A node in either state (i) or (ii) is said to be *fathomed*.

A node that has not yet been fathomed is said to be *active*.

A node in states (i) or (iib) cannot contain the optimum, it is *pruned* from the tree.

* *incumbent* = best integer solution found so far

Branch and Bound - main steps

The search continues, branching from the active nodes until either :

(i) all nodes have been pruned from the tree

(in which case the problem has no feasible solutions)

(ii) all active nodes have bounds no greater than the value of the incumbent

(in which case the incumbent represents an optimal solution)

Branch and Bound for KP - principles

- The solution set is partitioned into subsets $S_1, \ldots, S_M$
  each subset corresponds to a partial (feasible) solution,
  i.e. some of the variables have been fixed
- When a *branching* with respect to a variable x_j is done, 2 new subsets are created from S_k :
  S_k^0 where we set $x_j = 0$, S_k^1 where we set $x_j = 1$
- An upper bound UB_k corresponding to S_k can be computed by the greedy algorithm on the CKP corresponding to S_k (integer part)
- A lower bound LB_k corresponding to S_k can be computed by rounding down the fractional variable, if it exists.
- If $UB_k = LB_k$ the subproblem k is solved exactly.

Branch and Bound for KP - principles

When we continue branching ?

- Let z^* be the value of the best feasible solution found so far:
  current best solution: $z^* \geq \max\{LB_k, \ \forall k\}$
- $UB_k > z^*$: the subset S_k may contain a feasible solution with value $> z^*$.
  We continue working on S_k. (S_k, UB_k) is *active* $\Rightarrow$ *Branching*
- $UB_k \leq z^*$: the subset S_k cannot contain a solution with value $> z^*$.
  The subset is of no interest anymore. $\Rightarrow$ *Fathoming*

Branch and Bound for KP - steps

Initialization

- construct a list L containing all active pairs (S_k, UB_k) :
  initially, the original problem
- initial Upper Bound UB : solution of the CKP via the greedy algorithm
- initial Lower Bound LB : a feasible solution
  (rounding down the value of the fractional variable in the solution of CKP)

Branch and Bound for KP - steps

As long as L is not empty :

- choose among the active pairs the one with the maximal UB : (S_k, UB_k)
- split S_k in S_k^0 and S_k^1 according to a branching criterion
  (ex. : the fractional variable x_i in the continuous solution)
- compute UB_k^0 and UB_k^1 (greedy algorithm for CNP)
- compute LB_k^0 and LB_k^1 (feasible solution for KP)
- if $LB_k^0 > z^*$, then set $z^* \leftarrow LB_k^0$ and update x^* accordingly
  if $LB_k^1 > z^*$, then set $z^* \leftarrow LB_k^1$ and update x^* accordingly
- if $UB_k^0 > z^*$, then add (S_k^0, UB_k^0) to L
  if $UB_k^1 > z^*$, then add (S_k^1, UB_k^1) to L
- update L deleting the elements (S_h, UB_h) such that $UB_h < z^*$

Branch and Bound for KP - algorithm

Sort values/volumes by non-increasing order.
Let i_j be indexes such that u_{i_j}/v_{i_j} with $j = 1$ is the maximum ratio

$x_i \leftarrow 2 \quad \forall i$

$k \leftarrow 1$ // iteration counter

$L \leftarrow (S_k, UB_k)$ // original problem

$z^* \leftarrow LB_k$

while (L not empty) **do**

  $\nu \leftarrow \min\{i_j | j \in \{1, ..n\} \wedge x_\nu = 2\}$

  $x^0_\nu \leftarrow 0, \quad x^1_\nu \leftarrow 1$ // two branches

  for $p = 0, 1$ **do**

    compute UB^p_k and LB^p_k

    if $\left(\sum_{j=1}^{\nu} v_{i_j} \leq V\right)$ **then**

      if $(UB^p_k > z^*$ and $LB^p_k \neq UB^p_k)$ **then**

        add (S^p_k, UB^p_k) to L by non-increasing order for UB

Branch and Bound for KP - algorithm (cont.)

      if $(LB^p_k > z^*)$ **then**

        $z^* \leftarrow LB^p_k$ // current best solution

        $x^* \leftarrow$ corresponding solution

      end if

  end for

  $L \leftarrow L \setminus (S_k, UB_k)$

  $k \leftarrow k + 1$

end while

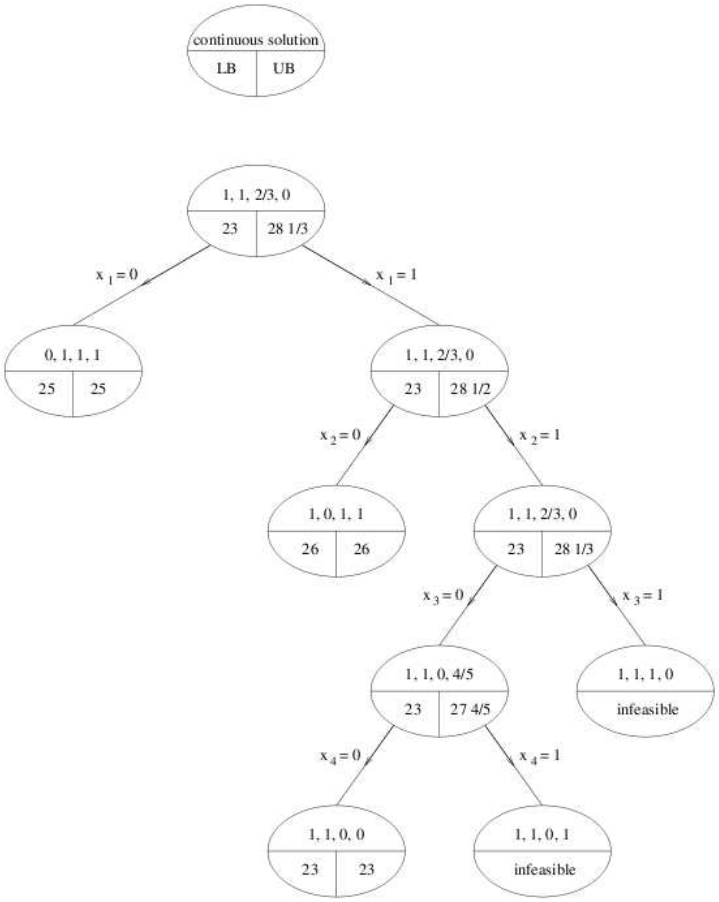
Example

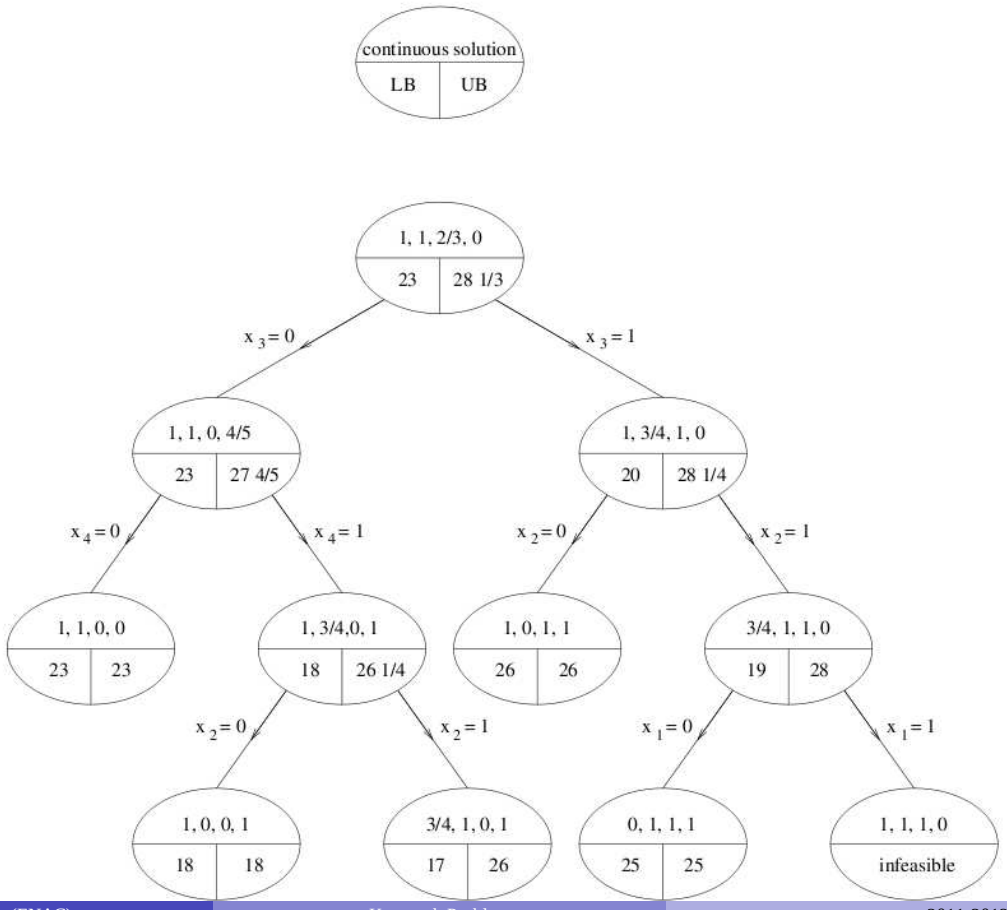
$$\begin{cases} \max & z = 12x_1 + 11x - 2 + 8x_3 + 6x_4 \\ \text{s.t.} & 8x_1 + 8x_2 + 6x_3 + 5x_4 \leq 20 \\ & x_i \in \{0,1\}\ \forall i \end{cases}$$

$$\frac{u_1}{v_1} > \frac{u_2}{v_2} > \frac{u_3}{v_3} > \frac{u_4}{v_4}$$

branching on the variable with lowest index

www.win.tue.nl/ wscor/OW/2V300





Designing a Branch and Bound algorithm

Branch and Bound is a very general framework.

To completely specify how the process is to proceed, you also need to define

- How to obtain bounds?
- How to branch?
- In what order to examine nodes?

Branch and Bound: Obtaining bounds

For a maximization problem

- the (current) tightest lower bound, provided by the best solution we have found, is also referred to as the *primal bound*.
- the upper bound on the profit obtainable within a given set of solutions is also referred to as the *dual bound*.

- To obtain a dual (upper) bound :

  one typically solves a relaxation of the subproblem :

  LP relaxation (but also Lagrangian relaxation, etc..)

  The bound can also be computed by solving the *dual problem*.

- To obtain a primal (lower) bound :

  one typically computes a feasible solution of the subproblem.

Dual Problem

For a LP problem in standard Primal form:

$$(LP) \quad \begin{cases} \max & c^T x & x \in \mathbb{R}^n, c \in \mathbb{R}^n \\ \text{s.t.} & Ax \leq b & A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m \\ & x \geq 0 \end{cases}$$

The Dual Problem is :

$$(D) \quad \begin{cases} \min & b^T y & y \in \mathbb{R}^m \\ \text{s.t.} & A^T y \geq c \\ & y \geq 0 \end{cases}$$

Dual Problem

To obtain the Dual Problem:

- write the Lagrangian : $L(x,y,s) = c^T x - s^T x + y^T(b - Ax)$, with $s \in \mathbb{R}^n, y \in \mathbb{R}^m$
  (using nonnegative Lagrangian multipliers to add the constraints to the objective function)
- write the Lagrangian dual : $\min \max(yb + (c^T - s - yA)x)$
- apply the optimality conditions (KKT) to the Lagrangian dual

The solution of the dual problem provides an upper bound
to the solution of the primal problem

Dual Problem : Example

Knapsack with 2 constraints, relaxed

$$\begin{cases} \max & \sum_{i=1}^{n} x_i u_i \\ \text{s.t.} & \sum_{i=1}^{n} x_i v_i \leq V \\ & \sum_{i=1}^{n} x_i p_i \leq P \\ & x_i \leq 1 \ \forall i \\ & x_i \geq 0 \ \forall i \end{cases}$$

$$A = \begin{pmatrix} v_1 & \dots & v_n \\ p_1 & \dots & p_n \\ 1 & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & 1 \end{pmatrix} \quad b = \begin{pmatrix} V \\ P \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

Dual Problem : Example

Knapsack with 2 constraints, relaxed

$$A^T = \begin{pmatrix} v_1 & p_1 & 1 & \dots & 0 \\ \vdots & \vdots & 0 & \dots & 0 \\ v_n & p_n & 0 & \dots & 1 \end{pmatrix}$$

$$(D) \quad \begin{cases} \min & b^T y = \sum_{i=1}^{n+2} b_i y_i \\ \text{s.t.} & A^T y \geq u \\ & y \geq 0 \end{cases} \qquad \Leftrightarrow \; v_i y_1 + p_i y_2 + y_{i+2} \geq u_i \quad \forall i \in \{1, ..n\}$$

Dual Problem : Example

Knapsack with 2 constraints, relaxed

A (feasible) solution can be obtained by setting :

$$y_1 = y_2 = 0, \quad y_{i+2} = u_i \; \forall i$$

The solution is : $\sum_{i=1}^{n} u_i$. *This gives an upper bound for the 0-1 KP.*

Another solution can be obtained by setting :

$$y_1 = y_2 = 1, \quad y_{i+2} = \max\{0, u_i - p_i - v_i \; \forall i\}$$

The solution is :

$$V + P + \sum_{i=1}^{n} \max\{0, u_i - p_i - v_i\}$$

that can be better than the previous one if $V + P < \sum_{i=1}^{n} u_i$.

So, we can search for

$$\min_{\alpha \in \mathbb{R}^+} \alpha(V + P) + \max\{0, u_i - \alpha(p_i + v_i)\}$$

Branch and Bound: How to partition into subproblems

- IP problems :

  - choose $x_j \notin \mathbb{Z}$
  - set $S_k^j = S_k \cup \{x : x_j \leq \lfloor x_j^k \rfloor\}$ and
    $S_k^{j+1} = S_k \cup \{x : x_j \leq \lfloor x_j^k \rfloor + 1\}$

  How to choose the fractionary component? Example : choose the component with fractionary value closest to 0.5

- Binary IP problems :

  - choose x_j
  - set $S_k^0 = S_k \cup \{x : x_j = 0\}$ and
    $S_k^1 = S_k \cup \{x : x_j = 1\}$

Branch and Bound: How to choose a node

The search tree can be explored using

- **Depth-first search**
  allows to find a feasible solution to the IP quickly
- **Breadth-first-search**
  may allow to prune more of the search tree sooner
- **Best-Node First**
  chooses the node with the best dual bound.
  This might lead to finding a good incumbent quickly, thus eliminating the need to evaluate the other subproblems
- Other **heuristics**

Bibliography I

L.R. Foulds.
Combinatorial Optimization for Undergraduates
Springer-Verlag, 1984.

C.H. Papadimitriou and K. Steiglitz.
Combinatorial Optimization: Algorithms and Complexity.
Dover, New York, 1998.

John W. Chinneck.
Practical Optimization: a gentle introduction
`www.sce.carleton.ca/faculty/chinneck/po.html`, 2010.