# M2 ORO: Advanced Integer Programming

Sophie Demassey

Mines Nantes - TASC - INRIA/LINA CNRS UMR 6241
sophie.demassey@mines-nantes.fr

October 10, 2011





Part IV

## Solving MILP (1)





## Outline







## easy IP

There are a number of combinatorial optimization problems for which the convex hull of the solution is explicit and of polynomial size.
Examples:

- network flow problems: transhipment problem, assignment problem, matching problem, shortest path problem, etc.
- IP whose matrix of constraints is totally unimodular

These are *easy* problems:

- they belong to the class of complexity $\mathcal{P}$
- the optima of the LP relaxation are integer
- the optima of the IP are the optima of the LP relaxation

## how to solve easy IP ?

Given an ideal IP formulation, an easy problem can be solved by one of the following methods:

- solving the LP relaxation using the simplex algorithm
- solving the LP relaxation using a polynomial-time interior point algorithm (ellipsoid method is not used in practice)
- using a dedicated algorithm: polynomial-time algorithm for network problems (ex: Dijsktra for the shortest path, Ford-Fulkerson for max-flow, Hungarian method for assignment)

**Which method to choose ?**

- a generic LP algorithm is usually not as efficient as a dedicated algorithm with low complexity
- but is may be easier to create an IP model and to solve it with any available LP solver rather than to implement a dedicated algorithm





## hard IP

What can we do if:

- the IP formulation is not ideal: $conv(S) \subsetneq LP(S)$
- or if the IP formulation is of exponential size ?

Two methods for solving hard IP:

- cutting-plane methods
- LP-based branch-and-bound methods

Obviously these methods do not run in polynomial time in the general case.





## Outline

1 easy problems

2 cutting-plane methods
- definitions
- examples

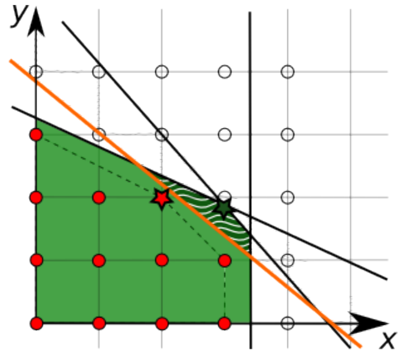




## cutting-plane method

**the idea:**

strengthening an IP formulation by adding constraints iteratively to get $LP(S)$ as close as possible to $conv(S)$



## cutting-plane method

This approach applies to any optimization problem $(P) : max\{cx \mid x \in S\}$ with any LP relaxation $LP(P) : max\{cx \mid Ax \leq B\}$ with $S \subseteq \{x \mid Ax \leq b\}$.

**Dantzig-Fulkerson-Johnson (TSP, 1954)**

1. compute an optimum solution $\bar{x}$ of $LP(P)$
2. if $\bar{x} \in S$ then STOP because $\bar{x}$ is optimum for $(P)$
3. otherwise find one or several linear inequalities separating $\bar{x}$ from $S$
4. update $LP(P)$ by adding them to system $Ax \leq b$ and goto 1.

- step 3 is called the separation problem
- it is generally as hard as the problem itself
- unless we search for a specific family of inequalities, called a template





## Definitions

Let $(P) : max\{cx \mid x \in S \cap \mathbb{Z}^n_+\}$ an IP defined on polyhedron $S = \{x \in \mathbb{R}^n_+ \mid Ax \leq b\}$ and $(\bar{P})$ the LP relaxation

- a valid inequality for $(P)$ is any linear inequality $\pi x \leq \pi_0$ that is satisfied by any feasible solution of $(P)$:

$$\pi x \leq \pi_0 \quad (\forall x \in S \cap \mathbb{Z}^n_+)$$

- let $\bar{x}$ be an optimum solution of $(\bar{P})$, a valid inequality $(\pi, \pi_0) \in \mathbb{R}^n \times \mathbb{R}$ for $(P)$ is a cutting-plane if it is violated by $\bar{x}$:

$$\pi x \leq \pi_0 \quad (\forall x \in S \cap \mathbb{Z}^n_+) \qquad \text{and} \qquad \pi \bar{x} > \pi_0$$





## Example 1: Mixed Integer Rounding Cuts

Combining constraints, then rounding leads to valid inequalities.

Let $u \in \mathbb{R}^m_+$, then the following inequalities are valid for $(P)$:

- surrogate: $\sum_{j=1}^m u_j a_{ij} x_i \leq \sum_{j=1}^m u_j b_j$ (since $u \geq 0$)
- round off: $\sum_{j=1}^m \lfloor u_j a_{ij} \rfloor x_i \leq \sum_{j=1}^m u_j b_j$ (since $\lfloor u_j a_{ij} \rfloor \leq u_j a_{ij}$ and $x \geq 0$)
- Chvátal-Gomory: $\sum_{j=1}^m \lfloor u_j a_{ij} \rfloor x_i \leq \lfloor \sum_{j=1}^m u_j b_j \rfloor$ (since $e \in \mathbb{Z}$ and $e \leq f$ implies that $e \leq \lfloor f \rfloor$)

- CG inequalities form a generic class of valid inequalities: they apply to any IP
- conversely, we can prove that any valid inequality for any IP is of that kind !





## Chvátal-Gomory procedure

**Chvátal-Gomory procedure**

1. compute the optimal solution $\bar{x}$ of $(\bar{x})$
2. find $u > 0$ s.t. $\lfloor ua \rfloor \bar{x} > \lfloor ub \rfloor$, STOP otherwise
3. add constraint $\lfloor ua \rfloor x \leq \lfloor ub \rfloor$ to $(\bar{P})$ then goto 1

- Theorem: every valid inequality of an IP can be obtained by applying Chvátal-Gomory procedure for a finite number (even exponential) of times.
- separation oracle: different systematic ways to choose $u$
- for ex, Gomory cut: compute $u$ from the optimal simplex basis

## Generic vs. Specific templates

- surrogate, zero-half, CG, MIR cuts are fully generic templates
- some templates are problem-specific, for ex: odd-set inequalities for matching, subtour elimination inequalities for TSP
- as intermediary, some other templates are generic for families of MIP sharing a given structure





## Example 2: Clique Cuts

**A typical MIP structure**

$$\begin{aligned} &\max cx + c'y && \\ &\text{s.t. } Ax + A'y \leq b && (1)\\ &\quad \Delta y \leq 1 && (2)\\ &\quad x \in \mathbb{R}^m_+, y \in \{0,1\}^n && \end{aligned}$$

$\Delta$ is a matrix of $0$ and $1$.

- in each constraint $(2)$ there is at most one $y_i = 1$
- build a graph $G$ with a vertex for each $i$ and an edge for each pair $(i, j)$ s.t. $y_i$ and $y_j$ appear both in a constraint $(2)$
- $\sum_{i \in C} y_i \leq 1$ is a valid inequality for each clique $C$ in $G$





## Clique Cuts Separation

- the strongest inequalities correspond to cliques of maximal cardinality
- but finding a maximum clique takes exponential time, ex: Tarjan&Trojanowski in $O(1.26^n)$
- finding a maximal clique is much easier but it may give a very small clique (and weak inequality)
- the inequality is a cut if $\sum_{i \in C} \bar{y}_i > 1$: search for a clique of maximal weight (NP-hard too)
- the clique cuts are then usually searched heuristically
- note that the graph is computed once at the beginning of the search, then filtered according to the fixed variables





## Structure-specific cuts

- there exist many other templates that apply to MIP with specific structure, e.g:
  - cover cuts $\sum_{i \in K} y_i \leq |K| - 1$: where $K$ is a minimal cover $\sum_{i \in K} a_i > b$ for $\sum_i a_i y_i \leq b$
  - GUB cuts: a cover cut for $\sum_i a_i y_i \leq b$ sharing at most one variable for each clique constraints $\sum_{i \in Q_j} y_i \leq 1$
  - disjunctive cuts for disjunctive problems: $P_1 \cap P_2 \subseteq S$ but $P_1 \cup P_2 \not\subseteq S$
  - flow cover, flow path, implied bound cuts for problems with continuous variables and UB indicator variables
- these structures can be automatically detected
- cuts are generated at a preprocessing step or on the fly in an iterative/incremental fashion
- generic and structure-specific cuts are fundamental ingredients of modern MIP solvers

## Exercices

**Cover inequalities**

Find a non-dominated cover inequality $\sum_{i \in K} y_i \leq |K| - 1$ for:

$$S = \{y \in \{0,1\}^7 | 11y_1 + 6y_2 + 6y_3 + 5y_4 + 5y_5 + 4y_6 + y_7 \leq 19\}$$

**GUB inequalities**

Find a non-dominated GUB inequality $\sum_{i \in C} y_i \leq |C| - 1$ for:

$$\begin{aligned} S = \{\ & y \in \{0,1\}^8 \\ \text{s.t. } & 2y_1 + y_2 + 5y_3 + 2y_4 + 3y_5 + 6y_6 + 4y_7 + y_8 \leq 9 \\ & y_1 + y_4 + y_6 \leq 1 \\ & y_5 + y_8 \leq 1 \\ & y_2 + y_7 \leq 1\ \} \end{aligned}$$





## Answer: Cover inequalities

**Cover inequalities**

$$S = \{y \in \{0,1\}^7 | 11y_1 + 6y_2 + 6y_3 + 5y_4 + 5y_5 + 4y_6 + y_7 \leq 19\}$$

- $(y_3, y_4, y_5, y_6)$ is a minimal cover for $11y_1 + 6y_2 + 6y_3 + 5y_4 + 5y_5 + 4y_6 + y_7 \leq 19$ as $6 + 5 + 5 + 4 > 19$ then $y_3 + y_4 + y_5 + y_6 \leq 3$ is a cover inequality
- we can derive a stronger valid inequality $y_1 + y_2 + y_3 + y_4 + y_5 + y_6 \leq 3$ by noting that $y_1, y_2$ has greater coefficients than any variable in the cover
- note furthermore that $(y_1, y_i, y_j)$ is a cover $\forall i \neq j \in \{2,3,4,5,6\}$ then $2y_1 + y_2 + y_3 + y_4 + y_5 + y_6 \leq 3$ is also valid

The procedure to get this last equality is called *lifting*





## Answer: GUB inequalities

**GUB inequalities**

$$\begin{aligned} & 2y_1 + y_2 + 5y_3 + 2y_4 + 3y_5 + 6y_6 + 4y_7 + y_8 \leq 9 \\ & y_1 + y_4 + y_6 \leq 1 \\ & y_5 + y_8 \leq 1 \\ & y_2 + y_7 \leq 1 \end{aligned}$$

- $(y_1, y_6, y_7)$ is a minimal cover having 2 variables in the first clique inequality, then the associated cover inequality $y_1 + y_6 + y_7 \leq 2$ is redundant with $y_1 + y_6 \leq y_1 + y_4 + y_6 \leq 1$
- $y_1 + y_5 + y_7 \leq 2$ is a GUB inequality, i.e. the cover has at most one variable in each clique constraint
- by lifting, we can strengthen it: $y_1 + y_5 + y_7 + y_3 + y_6 \leq 2$





## Separation with templates

problem-specific cuts are usually derived according to:

**the template paradigm**

1. describe one or more templates of linear inequalities that are satisfied by all the points of $S$
2. for each template, design an efficient separation algorithm that, given an $\bar{x}$, attempts to find a cut that matches the template.

The separation algorithm may be:

- exact: finds a cut that separates $\bar{z}$ from $S$ and matches the template whenever one exists
- heuristic: sometimes fails to find such a cut even though one exists.

## Example 3: Cuts for the Maximum Independent Set Problem

**Maximum Independent Set Problem**

Find a subset $S$ of pairwise non-adjacent vertices in a graph $G(V,E)$ of maximum cardinality.

- model as an IP
- show that $\bar{z}/z^*$ may be arbitrary large





## Maximum Independent Set

**Maximum Independent Set Problem**

$$\begin{aligned} \max \sum_{i \in V} x_i & \\ \text{s.t. } x_i + x_j \leq 1 & \qquad (i,j) \in E \\ x_i \in \{0,1\} & \qquad i \in V \end{aligned}$$

- $x_i = 1$ iff $i$ belongs to the independent set
- *note that the matrix is not TU in the general case*
- For a complete graph $G = \mathbb{K}^n$:
  $\bar{z} = n/2$ (with $x_i = 1/2$ for each vertex $i$), but $z^* = 1$





## Odd-Cycle Inequalities

- any odd cycle $C$ of $G$ has at most $\frac{|C|-1}{2}$ independent vertices
- $\sum_{i \in C} x_i \leq \frac{|C|-1}{2}$ is a valid inequality
- with this set of constraints the formulation can be stronger (prove it)
- but it is not ideal (prove it)





## Odd-Cycle Inequalities

$$\begin{aligned} (P') : \max \sum_{i \in V} x_i & \\ \text{s.t. } x_i + x_j \leq 1 & \qquad (i,j) \in E \\ \sum_{i \in C} x_i \leq \frac{|C|-1}{2} & \qquad C \text{ odd cycle} \\ x_i \in \{0,1\} & \qquad i \in V \end{aligned}$$

- for $G = \mathbb{K}^4$ and $x_i = 1/3\ \forall i$ we get: $\bar{z}' = 4/3$, then
- $1 = z^* < \bar{z}' < \bar{z} = 2$

## Odd-Cycle Separation

### How to generate an Odd-Cycle cut ?

- the odd-cycle inequality is equivalent to $\sum_{(i,j)\in E(C)} y_{ij} \geq 1$ with *distance* variables $y_{ij} = 1 - x_i - x_j$
- we search for the minimum length $d$ of an odd cycle in $G$ with distance $d_{ij} = 1 - \bar{x}_i - \bar{x}_j \geq 0$ (how ?)
- if $d \geq 1$ there is no currently violated odd-cycle inequality
- otherwise generate the cut associated to the minimum length cycle
- this constraint cannot already be present in the model as it is violated by $\bar{x}$
- the procedure always stops in a finite number of iterations





## Odd-Cycle Separation

### How to find a shortest odd cycle ?

- build an undirected bipartite graph $G'$ from $G$ by duplicating the vertices $V$ and each edge $(i,j)$ as $(i_1, j_2)$ and $(j_1, i_2)$
- each odd cycle $C$ in $G$ with $i \in C$ corresponds to a path from $i_1$ to $i_2$ in $G'$
- then a shortest odd cycle of $G$ can be computed in polynomial time by finding a shortest path in $G'$





## Example 4: Cuts for TSP

### TSP

$$\min \sum_{e\in E} c_e x_e \tag{1}$$

$$\text{s.t.} \sum_{e\in E | i\in e} x_e = 2 \qquad i \in V \tag{2}$$

$$\sum_{\delta(Q)} x_e \geq 2 \qquad \emptyset \subsetneq Q \subsetneq V \tag{3}$$

$$x_e \in \{0,1\} \qquad e \in E \tag{4}$$

- contraints (3) are the subtour elimination inequalities where $\delta(Q)$ is the cutset of $Q$





## Example 4: Cuts for TSP

- Dantzig-Fulkerson-Johnson (1954) solved a 49 cities instance by manually and dynamically finding subtour cuts
- there is an exponential number of subtour contraints but the iterative approach avoids to generate them all
- Martin's template paradigm (1966): if $\bar{x}$ is fractional then a Gomory cut exists, otherwise each proper connected component of the graph formed by $\bar{x}$ induces a subtour cut
- templates of hypergraph inequalities are nowaday used instead of Gomory cuts: blossom (Edmonds, 1965) and comb (Grötschel, 1980)
- many theoretical and computational improvements were brought by Applegate, Bixby, Chvátal, Cook in the Concorde code (1994-2000)

## Cutting-plane methods

- cuts may either be generic, structure-specific, or problem-specific
- separation oracle: exact/heuristic algorithm for finding cuts
- the whole method terminates if exact separation of a finite family of inequalities





## Limits of cutting-plane methods

- adding cuts changes the IP structure and may complicate its solution
- even if it terminates, the method may take a very long time !
- to design an ad-hoc separation algorithm is an hard task
- the LP solver may be unable to process the whole set of generated cuts
- to design a data structure able to store and manage the cuts is an hard task





## Limits of cutting-plane methods

- cuts help at improving the relaxation bound
- the improvements get smaller as more cuts are added
- when they become unbearably small, the sensible thing to do is to branch





## Some references


- *Cutting planes in integer and mixed integer programming* (2002)
  H. Marchand, A. Martin, R. Weismantel and L.A. Wolsey, Disc. App. Math. 123/124:391–440. (survey)
- *Computational Integer Programming and Cutting Planes* (2005)
  A. Fügenschuh and A. Martin, Disc. Opt. 12:69-121. (cuts in modern solvers)
- *Valid inequalities for mixed integer linear programs* (2008)
  G. Cornuéjols, Math. Prog. 112(1):3-44. (MIR cuts, lifting, polyhedra)
- *TSP cuts which do not conform to the template paradigm* (2001)
  D. Applegate, R Bixby, V Chvátal and W Cook, Comp. Comb., Opt. LNCS 2241:261-303. (cuts for TSP, computation, beyond template)
- *Lifted cover inequalities for 0-1 integer programs: Computation* (1998)
  Z. Gu, G.L. Nemhauser and M.W.P. Savelsbergh, IJOC, 10:427-437. (cuts for Knapsack: cover, GUB, lifting)
- *Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems* (2003) D. Applegate, R Bixby, V Chvátal and W Cook, Math. Prog. 97:91–153. (cuts for large-scale applications)

Part V

## Solving MILP with LP-based Branch-and-Bound





## Today's lecture:







## Outline







## from the LP relaxation to an IP solution

if an IP is of reasonable size

$$(P) : z = \max\{cx \mid x \in S\} \text{ with } S = \{x \in Z^n_+ \mid Ax \leq b\}$$

then its LP-relaxation can be solved in reasonable time

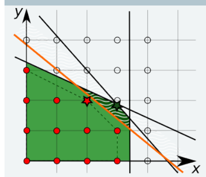
$$(\bar{P}) : \bar{z} = \max\{cx \mid x \in \bar{S}\} \text{ with } \bar{S} = \{x \in R^n_+ \mid Ax \leq b\}$$

- if $(\bar{P})$ is unbounded and $S \neq \emptyset$ then $(P)$ is unbounded
- if $(\bar{P})$ is infeasible then $(P)$ is infeasible
- if $(\bar{P})$ optimum solution $\bar{x}$ is integer then $(P)$ optimum solution is $\bar{x}$
- otherwise ? an upper bound + a good non-feasible solution

## from LP relaxation to IP solution

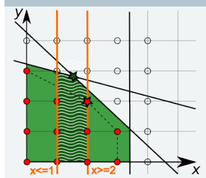
how to solve $(P)$ when the formulation is not ideal $conv(S) \subsetneq \bar{S}$ ?

**cutting-plane algorithm**




shrink progressively the search space by strenghtening the formulation

**branch-and-bound algorithm**




divide the search space and optimize recursively on every subspaces





## Outline





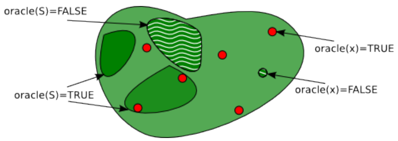


## B&B is an implicit enumeration method

**implicit enumeration**

a generic method for solving combinatorial satisfaction/optimization problems for which an oracle exists.
Let $\bar{S}$ the search space:

- oracle($x$) = FALSE if solution $x \in \bar{S}$ is not feasible
- oracle($U$) = TRUE if subspace $U \subseteq \bar{S}$ *may* have a feas/opt solution




if oracle(U) is evaluated to FALSE, then prune U from the search





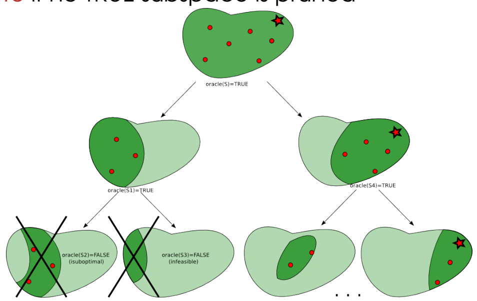
## B&B is a tree search method

implicit enumeration is complete if no TRUE subspace is pruned

**tree search**

divide $\bar{S}$ recursively
evaluate/prune subspaces

search structure = tree
$\bar{S}$ = root
subspace = node
pruned subspace = leaf



1. initialize $\mathcal{L} = \{\bar{S}\}$ (candidates)
2. choose $U$ in $\mathcal{L}$
3. if oracle$(U)$ = TRUE and $|\bar{S}'| > 1$, then divide $U = U_1 \cup \ldots \cup U_k$
4. remove $U$ from $\mathcal{L}$, add $U_1, \ldots, U_k$ to $\mathcal{L}$

## implicit enumeration + tree search

**examples**

| problem | algorithm | oracle |
|---|---|---|
| satisfaction | backtracking | feasibility checker |
| optimization | branch-and-bound | relaxation solver |
| MILP | LP-branch-and-bound | LP-relaxation solver |

- $z^*$ the max value (incumbent) found so far
- oracle$(U)$ = TRUE iff $\max\{cx | x \in U\} = c\bar{x}^U \geq z^* + 1$
- update $z^*$ if oracle$(U)$ = TRUE and if $\bar{x}^U \in S$





## Outline







## branch-and-bound for MIP

**branch-and-bound algorithm**

1. $\mathcal{L} = \{\bar{S}\}$
2. *choose* $U$, remove from $\mathcal{L}$
3. *evaluate* $U$, then:
   either *prune* $U$
   or *divide* $U$, add to $\mathcal{L}$
4. if $\mathcal{L} = \emptyset$ STOP
   otherwise GOTO 2

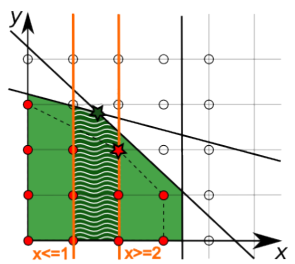
**a standard LP-based B&B for $(P): z = \max\{cx \mid Ax \leq b, x \in Z^n_+\}$**

- $\bar{S} = \{x \mid Ax \leq b\}$
- choose subspaces by LIFO (Depth-First Search)
- evaluate $U$: $\bar{z}^U = \max\{cx | x \in U\}$
  prune $U$ if $\bar{z}^U < z^* + 1$
  otherwise update $z^* = \bar{z}^U$ if $\bar{x}^U \in S$
  otherwise divide $U$ to exclude $\bar{x}^U$:
  $U \cap \{x_i \leq \lfloor \bar{x}^U_i \rfloor\}$ / $U \cap \{x_i \geq \lceil \bar{x}^U_i \rceil\}$
  for some variable index $i$





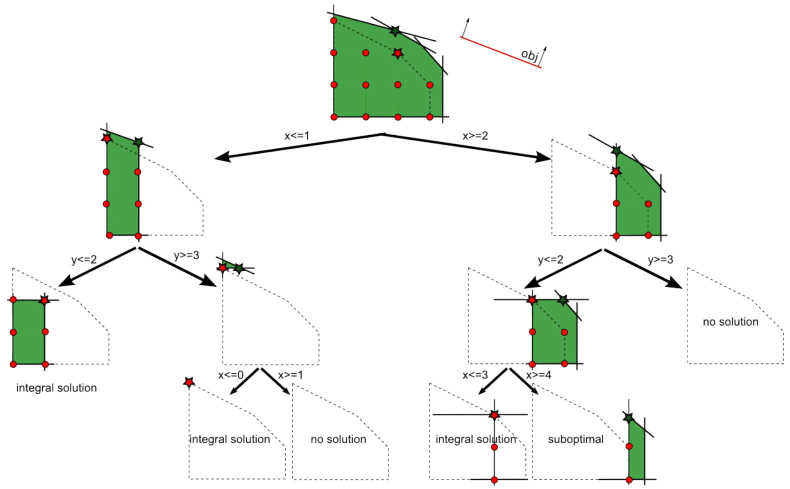
## branch-and-bound for MIP




**a standard LP-based B&B for $(P): z = \max\{cx \mid Ax \leq b, x \in Z^n_+\}$**

- $\bar{S} = \{x \mid Ax \leq b\}$
- choose subspaces by LIFO (Depth-First Search)
- evaluate $U$: $\bar{z}^U = \max\{cx | x \in U\}$
  prune $U$ if $\bar{z}^U < z^* + 1$
  otherwise update $z^* = \bar{z}^U$ if $\bar{x}^U \in S$
  otherwise divide $U$ to exclude $\bar{x}^U$:
  $U \cap \{x_i \leq \lfloor \bar{x}^U_i \rfloor\}$ / $U \cap \{x_i \geq \lceil \bar{x}^U_i \rceil\}$
  for some variable index $i$

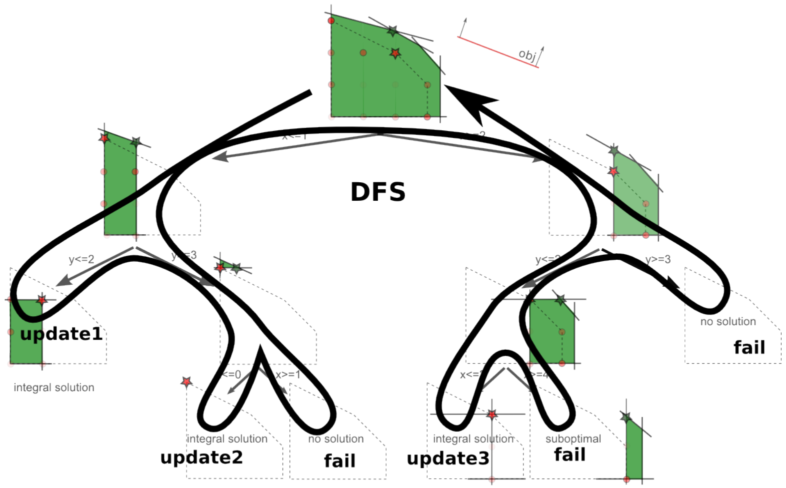
## LP-based branch-and-bound: example








## LP-based branch-and-bound: example








## search tree

- branch-and-bound draws a search tree or decision tree where each node corresponds to an evaluated subspace
- the initial search space $\bar{S}$ is at the root node
- the subspaces created by dividing a space $U$ are the child nodes of the parent node corresponding to $U$
- the action of dividing is known as branching
- a leaf in the tree is a subspace that is pruned
- at one iteration of the B&B algorithm, the candidates in $\mathcal{L}$ correspond to active nodes in the tree: the parents of the active nodes have all been evaluated in previous iterations, while the children will be created in the next iterations.





## 4 components

**bounding**

how to evaluate a node ?

- LP relaxation
- combinatorial relaxation,
- lagrangian relaxation, etc.

**pruning**

when discard a node ?

- infeasible: $U = \emptyset$
- or suboptimal: $\bar{z}^U < z^* + 1$
- or new incumbent: $\bar{x}^U \in \mathbb{Z}^n$

**branching**

how to divide a node ?

- variable branching
- pseudocost/strong branching
- constraint branching

**selection**

how to order active nodes ?

- Depth First Search
- Best First Search
- Best Estimate/Projection,...

## branching strategies

GOAL: accelerate the search
MEAN: try to minimize the number of nodes to evaluate

**3 combined heuristics**

1. choose the way a subspace is divided
2. choose the element of division
3. choose the subspace to divide

in order to keep the evaluation easy





## how to divide ?

the division strategy must be compatible with the bounding strategy:

- exclude the current relaxed solution
- exclude no feasible solution
- not overload the relaxed model
- not modify its structure





## branching on variables

**example: variable dichotomy**

- let $\bar{x}$ the LP solution, $\bar{x} \notin \mathbb{Z}^n$
- choose a fractional variable $\bar{x}_i \notin \mathbb{Z}$
- divide in two by shrinking the bounds of the variable in the two child LPs: $x_i \leq \lfloor \bar{x}_i \rfloor$ (left branch) and $x_i \geq \lceil \bar{x}_i \rceil$ (right branch)

- variable dichotomy is compatible to any LP relaxation
- default branching strategy in most solvers
- other variable branching: fix variable value in each branch
  $x_i = v_i^1 \ \vee x_i = v_i^2 \ \vee x_i = v_i^3 \ \vee \cdots \vee x_i = v_i^{p_i}$





## branching on constraints

**example: GUB dichotomy**

- if $(P)$ contains a GUB constraint $\sum_C x_i = 1, x \in \{0,1\}^n$
- choose $C' \subseteq C$ s.t. $0 < \sum_{C'} \bar{x}_i < 1$
- create two child nodes by setting either $\sum_{C'} x_i = 0$ or $\sum_{C'} x_i = 1$

- enforced by fixing the variable values
- leads to more balanced search trees
- special case when $C$ is logically ordered: SOS1 branching

## branching on constraints

**SOS1 branching in a facility location problem**

choose a warehouse depending on its size/cost:

$$\begin{aligned} \text{COST} &= 100x_1 + 180x_2 + 320x_3 + 450x_4 + 600x_5 \\ \text{SIZE} &= 10x_1 + 20x_2 + 40x_3 + 60x_4 + 80x_5 \\ \text{(SOS1)} &: x_1 + x_2 + x_3 + x_4 + x_5 = 1 \end{aligned}$$

- let $\bar{x}_1 = 0.35$ and $\bar{x}_5 = 0.65$ in the LP solution then SIZE$= 55.5$
- choose $C' = \{1, 2, 3\}$ in order to model SIZE$\leq 40$ or SIZE$\geq 60$
- the branching point of the SOS $C$ is given by:

$$\arg\max\{a_j \mid a_j < \sum_{i \in C} a_i \bar{x}_i, j \in C\}.$$



from the LP relaxation to an IP solution
branch-and-bound
LP-based branch-and-bound
definition
branching strategies

## division strategies

**examples**

- variable dichotomy $x_i \in [0, a] \vee x_i \in [a + 1, b]$
- branching on semi-continuous variables $x_i \in \{0\} \cup [a, b]$
- branching on domain values $x_i = 0 \vee x_i = 1 \vee \cdots \vee x_i = u$
- GUB branching
- SOS1 branching
- SOS2 branching $x_1 + x_2 + x_3 + x_4 + x_5 = 1$, if $\bar{x}_2 = \bar{x}_4 = 0.5$ set either $x_4 = x_5 = 0$ (enforcing $x_2 > 0$) or $x_1 = x_2 = 0$ (enforcing $x_4 > 0$)
- branching on connectivity constraints in TSP, if $\sum_{e \in \delta(U)} \bar{x}_e = 2.5$ set either $\sum_{e \in \delta(U)} x_e = 2$ or $\sum_{e \in \delta(U)} x_e \geq 4$ (enforcing $\sum_{e \in \delta(U)} \bar{x}_e = 2k$)
- ... other problem specific strategies

Sophie Demassey, TASC, EMN-INRIA/LINA
Université de Nantes / M2 ORO / Advanced Integer Programming / L6: solving/B&B

from the LP relaxation to an IP solution
branch-and-bound
LP-based branch-and-bound
definition
branching strategies

## branching strategies

GOAL: accelerate the search
MEAN: try to minimize the number of nodes to evaluate

**3 combined heuristics**

1. choose the way a subspace is divided
2. choose the element (variable/constraint) of division
3. choose the subspace to divide

in order to keep the tree size small

Sophie Demassey, TASC, EMN-INRIA/LINA
Université de Nantes / M2 ORO / Advanced Integer Programming / L6: solving/B&B

from the LP relaxation to an IP solution
branch-and-bound
LP-based branch-and-bound
definition
branching strategies

## variable choice rule

**choosing the fractional variable in variable dichotomy ?**

- choose the most infeasible variable: with the most fractional value (closest to $1/2$)
- or choose the *most suboptimal* variable: that causes the LP optimum to deteriorate quickly

- the first strategy aims at fixing the *hesitating* variables: often as good as a random choice
- the second strategy is the most usual in solvers
- helps in keeping the tree size small by augmenting pruning (when $\bar{z} < z^* + 1$)
- but it can be too expensive to compute the optimum changes for each fractional variable at each node

Sophie Demassey, TASC, EMN-INRIA/LINA
Université de Nantes / M2 ORO / Advanced Integer Programming / L6: solving/B&B

## estimates the quality of branching

- it is measured by the change in the objective function of the children $\bar{S}_i^-$ (left) and $\bar{S}_i^+$ (right) compared to the parent node $\bar{S}$:

$$score(x_i) = (1-\mu)\min(\bar{z}-\bar{z}_i^-, \bar{z}-\bar{z}_i^+) + \mu\max(\bar{z}-\bar{z}_i^-, \bar{z}-\bar{z}_i^+)$$

- how to estimate the cost of forcing $x_i$ to become interger ?





## estimates the quality of branching

- full strong branching: compute exactly $\bar{z}_i^-$ and $\bar{z}_i^+$ for each candidate $x_i$
- strong branching: consider only a subset of candidates and estimates (UB of) $\bar{z}_i^-$ and $\bar{z}_i^+$ by running only few dual simplex iterations
  - select candidates by considering: their infeasibility degree (the most fractional value) or their coefficient weight in the objective function
- pseudocost branching: keep an history of the successes of each variable in previous branchings
- solvers often allow more time to these expensive strategies at the top of the search tree





## problem-specific variable ordering

**example: a capacitated facility location problem**

each customer demand has to be fully satisfied from a single facility:
$y_i = 1$ if a facility is located at site $i$
$x_{ij} = 1$ if customer $j$ is served from facility $i$

- decisions on $y$ affect the overall solution more than decisions on $x$
- branch first on $y$

**problem/structure-specific branching strategies**

- select the most decisive variables first
- usually more efficient than general purpose strategies





## branching strategies

GOAL: accelerate the search
MEAN: try to minimize the number of nodes to evaluate

**3 combined heuristics**

1. choose the way a subspace is divided
2. choose the element of division
3. choose the subspace to divide

in order to find good feasible solutions early

## node selection strategies

### Depth First Search

choose one of the two newly created nodes

- descends quickly to find a first feasible solution (and incumbent $z^*$)
- allows incremental run of the simplex
- keeps the number of active nodes small

### Best First Search

choose the node with the best (largest upper) bound

- minimizes the total number of evaluated nodes (never selects a node whose UB is less than the optimum)
- diversifies the search by exploring more promising regions

### Best Estimate/Best Projection

choose the node with the best estimated optimum feasible solution





## 4 components

### bounding

how to evaluate a node ?
need a tight (LP-)relaxation to improve pruning

### pruning

when to discard a node ?
need good feasible solutions to improve pruning

### branching

how to divide a node ?
compatible with the relaxation
use its informations or problem knowledge to force early failures and keep the tree small

### selection

how to order active nodes ?
mix intensification (DFS) and diversification (BFS) to find good feasible solutions





## to go further

### other ingredients of a B&B

- preprocessing: tighten the LP relaxation at the root to improve bounding in every nodes
- terminaison: stop when $\mathcal{L} = \emptyset$ (B&B as an exact method) or before (B&B as a heuristic method)
- primal heuristic:
  - at the root node: get an good initial incumbent $z^*$ to inhibit growth of the tree
  - LP-based heuristic at every search nodes: continuously improve the best solution found so far
- dominance: prune a node that has no better descendant than another node
- branch-and-cut: generate cuts at each node that is not pruned
- branch-and-price: generate variables at each node





## Some references

- *Progress in linear programming-based algorithms for integer programming: an exposition* (2000) E.L. Johnson, G.L. Nemhauser and M.W.P. Savelsbergh, IJOC 12(1):2-23. (solvers survey)
- *Integer-programming software systems* (2005)
  A. Atamtürk and M.W.P. Savelsbergh, AOR 140(1):67-124. (solvers survey)
- *A computational study of search strategies for mixed integer programming* (1997)
  J.T. Linderoth and M.W.P. Savelsbergh, IJOC 11:173–187. (survey)
- *Branching rules revisited* (2004) T. Achterberg, T. Koch and A. Martin, OR Letters 33:42-54. (variable ordering)
- *Preprocessing and probing techniques for mixed integer programming problems* (1994) M.W.P Savelsbergh, ORSA JOC 6. (preprocessing)
- *Local branching* (2003)
  M. Fischetti and A. Lodi, Math. Prog. 98(1):23-47. (heuristic)
- *Parallel branch and cut* (2006)
  T.K. Ralphs, in Parallel Combinatorial Optimization (parallel B&B)
- *Pruning moves* (2010)
  M. Fischetti and D. Salvagnin, IJOC 22(1):108-119. (dominance)