Influence branching for learning to solve Mixed Integer Programs online

Paul Strang

Sureli Seminar
May 3, 2023

Introduction

Sommaire

Rappel du plan

Mixed Integer Programs

Mixed integer programs

Mixed integer linear programs are generally defined such as:

$$P : \begin{cases} \min c^T x \\ Ax \leq b \; ; \; x \in \mathbb{N}^{|\mathcal{I}|} \times \mathbb{R}^{n-|\mathcal{I}|} \end{cases}$$

with $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$

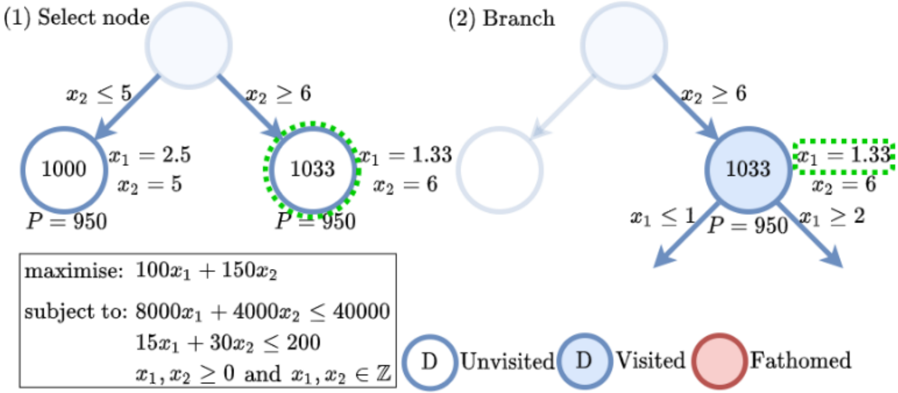
- Linear objective
- Linear constraints
- Integrity constraints makes the problem non-convex
- NP-hard in fact

Branch & bound

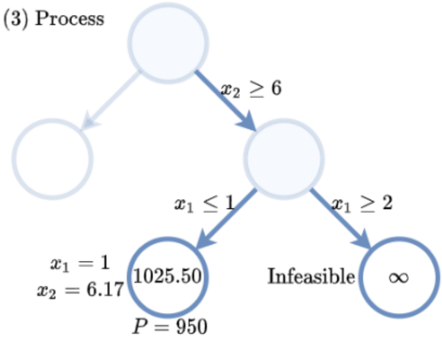
B&B :
Create a partition of the solution space by fixing binary variables at either 0 or 1 (branching).
Such partitioning is built following a tree structure, each node being a sub-problem of the initial MILP.
At each node, we solve the corresponding linear relaxation and hope to find a solution satisfying the binary constraints. A branch is expanded until we prune the leave nodes out of:

- Infeasibility
- Integrality
- Sub-optimality

Branch & bound

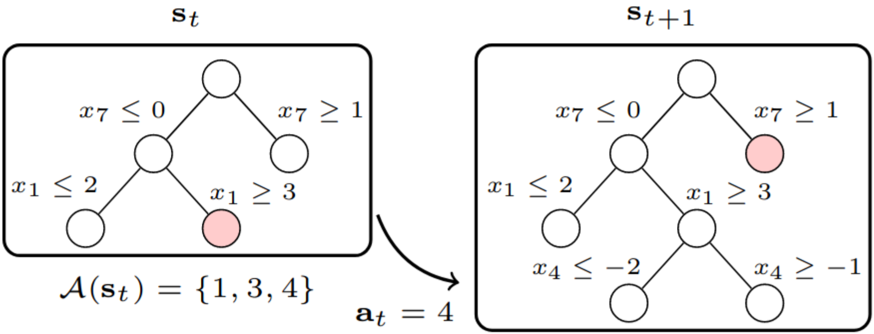


Branch & bound



Strong branching

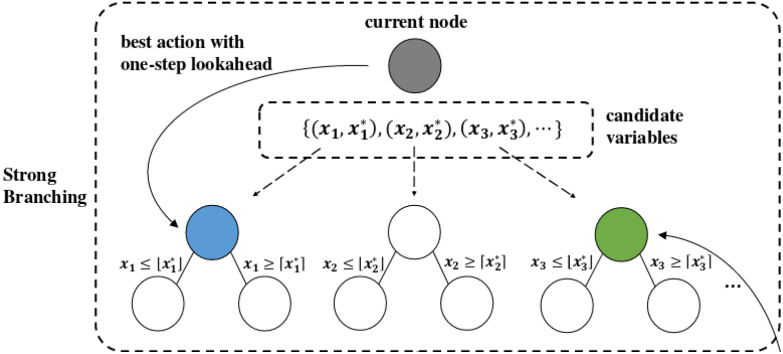
Two major selection strategies to parametrize the B&B solver :



The **node** selection strategy and the **variable** selection strategy !

Strong branching

Strong branching : select the best one-step lookahead branching in terms of **dual gap**



Inconvenient : vast number of LP iterations associated, intractable in most cases

Learning to solve MIPs

Industrial MIP solvers' node and variable selection strategies are based on complex fine-tuned heuristic, designed to perform best (in average) over a vast range of benchmarks.

In the context of real-world applications, in which similar instances with slightly varying inputs are solved on a regular basis, there is a huge incentive to reduce the solving time by learning efficient tailor-made heuristics.

MIP23 Computational competition

Each series $s \in \mathcal{S}$ of the competition if composed of 50 fixed-size MIPs sampled from an unknown distribution $\mathbb{Q}_s$.

$$i \in \mathcal{I}_s \sim \mathbb{Q}_s : \begin{cases} \min c^T x \\ Ax \leq b \ ; \ x \in \mathbb{N}^{|\mathcal{J}|} \times \mathbb{R}^{n-|\mathcal{J}|} \end{cases}$$

For each series, the value of one or several vectors among $\{A, b, c\}$ can change.

MIP23 Computational competition

- Series of 50 instances $i \in \mathcal{I}_s$ to solve **online** (sequentially).
- $t_{max} \sim 300\,s$ per instance
- $f_{s,\,i} = \frac{t}{t_{max}} + dualgap + no\ primal$
- $Total\ score = \sum_{s \in \mathcal{S}} \sum_{i=1}^{50} (1 + 0.1i) \cdot f_{s,\,i}$

MIP23 Computational competition

Trade-off between **learning a model** and **solving an instance** !

$\rightarrow$ Very different framework from the literature of machine learning applied to mixed integer programming.

$\rightarrow$ If you had to learn the absolute minimum about a MIP series, what would it be ?

Our take : for each series, we are going to try to learn graph representations of the instances leading to the best branching decisions near the root node of the B&B tree.

In fact, we introduce a new graph-based branching heuristic, named **influence branching**, and learn to fine tune it across instance series.

Rappel du plan

Definition

Local influence

We define the local influence w_{ij}^l exerted by variable i on variable j through constraint l. w_{ij}^l can be any function on A, b, c, in particular, we say that i has a non-zero influence on j through l if $\mathbb{1}_{A_{li} \neq 0} \mathbb{1}_{A_{lj} \neq 0} \neq 0$.

Direct influence

We define the direct influence w_{ij} exerted by variable i on variable j over P as :

$$w_{ij} = \mathbb{1}_{i \neq j} \sum_{l=1}^{m} w_{ij}^l$$

Definition

We can then derive a definition for influence graphs :

Influence graph

We call influence graph the directed graph $G = (V, E, W)$ where $V = \{1, ..., n\}$, $E = V \times V$ and where $W \in \mathbb{R}^{n \times n}$ the w_{ij} matrix satisfies the definition of direct influence.

Figure: Examples of influence graphs

Influence models

- **Count** $w_{ij}^{l} = \mathbb{1}_{A_{li}}\mathbb{1}_{A_{lj}}$
- **Binary** $w_{ij}^{l} = \frac{\mathbb{1}_{A_{li}}\mathbb{1}_{A_{lj}}}{\sum_{k=1}^{m}\mathbb{1}_{A_{ki}}\mathbb{1}_{A_{kj}}}$
- **Dual** $w_{ij}^{l} = \mathbb{1}_{A_{li}}\mathbb{1}_{A_{lj}}|y_l^*|$
- **Countdual**
  $w_{ij}^{l} = \mathbb{1}_{A_{li}}\mathbb{1}_{A_{lj}}\mathbb{1}_{(y_l{}^*\neq 0)}$
- **Auxiliary**
  $w_{ij}^{l} = \mathbb{1}_{A_{li}}\mathbb{1}_{A_{lj}}s_i|A_{li}y_l|$
- **Adversarial**
  $w_{ij}^{l} = \mathbb{1}_{A_{li}}\mathbb{1}_{A_{lj}}s_i|\frac{A_{li}}{A_{lj}}|\mathbb{1}_{(y_l{}^*\neq 0)}$

Table: Proposed influence models, with y^* the solution of the dual problem at the current node and s_i the minimal distance to a bound for variable i in the primal solution. $\mathbb{1}_{A_{li}\neq 0}$ is noted $\mathbb{1}_{A_{li}}$ to ease the notations.

Influence branching

Influence branching

The influence branching heuristic returns the variable within the graph with the maximal total influence :

$$w^* = \max_i w_i = \max_i \sqrt{1 + c_i} \sum_{j \neq i} w_{ij}(g) \tag{1}$$

as long as the depth of the current node d is inferior or equal to k, the maximum depth.

Influence branching is a variable selection strategy relying on two hyperparameters :

- $g \in \mathcal{G} = \{count, binary, ..., adversarial\}$, the influence model
- $k \in \mathbb{N}$, the maximal depth to apply the heuristic

Speed up potential on MIPcc23

Instance	Influence model	Max depth	Performance $f_{s,i}$	SCIP default	Speed up
1	binary	5	0.64	0.70	-0.06
2	adversarial	5	0.53	1.01	-0.48
3	countdual	5	0.49	0.60	-0.11
4	count	4	0.48	0.76	-0.28
5	countdual	2	0.70	1.03	-0.33
6	count	4	0.26	0.44	-0.18
7	auxiliary	3	0.42	0.53	-0.11
8	countdual	2	0.71	0.98	-0.37
9	countdual	4	0.57	1.39	-0.82
...	...	...	...	...	...
50	binary	5	0.29	0.66	-0.34
Avg			**0.56**	**0.94**	**-0.38**

Multi-armed bandit problem

- Learning which pair (g, k) performs best for any instance of any series would require to shift to a reinforcement learning framework
- We adopt an online bandits framework, as we try to learn which pair (g, k) obtains the best performance in average on a whole series of instances

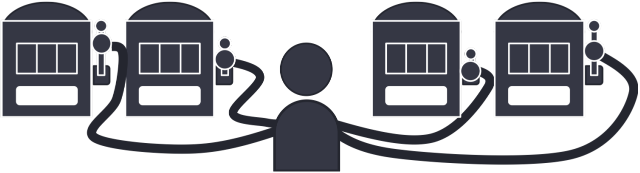
Multi-armed bandit problem

The optimization task can be reformulated as a multi-armed bandit problem on action space $\mathcal{A}$ where

$$\min_{a_i \in \mathcal{A}} \sum_{i=1}^{50} (1 + 0.1i)\, f_{s,\,i}(a_i) \tag{2}$$

is the sum of reward to minimize.

Multi-armed bandit problem



Multi-armed bandit problem

The optimization task can be reformulated as a multi-armed bandit problem on action space $\mathcal{A}$ where

$$\min_{a_i \in \mathcal{A}} \sum_{i=1}^{50} (1 + 0.1i)\, f_{s,\,i}(a_i) \tag{3}$$

is the sum of reward to minimize.

Multi-armed bandit problem

Influence model	Max depth	Performance	Speed up	Rank
count	5	0.857	**-0.0862**	**1**
base	6	0.865	-0.0783	2
countdual	2	0.874	-0.0691	3
base	5	0.877	-0.0657	4
count	4	0.882	-0.0606	5
...	...	...	...	...
adversarial	3	0.953	0.0520	34
adversarial	2	0.973	0.0721	35
auxiliary	5	1.05	0.148	36

Table: Sorted average performance of influence branching on *obj series 2* for each pair (g, k). The performance column corresponds to the mean of $f_{s,i} = \textit{reltime} + \textit{gap at time limit} + \textit{nofeas}$ over $\mathcal{I}_s$.

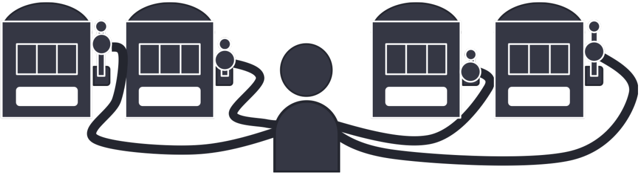
Rappel du plan

1. Mixed Integer Programming
2. Influence branching
3. Online learning
   - Building an action set for MIPcc23
   - Convergence
   - Results

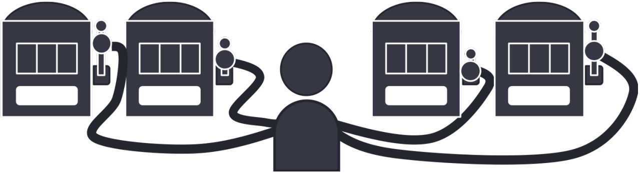
Building an action set

For each series, only 50 samples are available in total:

- Scores $\{f_{s,\,i}(a)\}_{i \in \mathcal{I}_s}$ with $a \in \mathcal{A} = \{(g, k) : g \in \mathcal{G},\ k \in [1, 6]\}$ are assumed to follow an unknown probability distribution $\mathcal{P}_{a,\,s}$.
- In order to minimize (3), the means of $(\mathcal{P}_{a,\,s})_{a \in \mathcal{A}}$, noted $(\mu_{a,\,s})_{a \in \mathcal{A}}$, need to be estimated (or at least ranked) as efficiently as possible for the heuristic to select the action leading to the minimum expected reward.



Building an action set



However :

- The more actions in the action space, the more samples are needed to guarantee the convergence of the bandits algorithm towards optimal actions.
- Moreover, the spreads between $(\mu_{a,\,s})_{a \in \mathcal{A}}$ are rather small, comprised between 0.01 and 0.2, in front of standard deviations of $(\mathcal{P}_{a,\,s})_{a \in \mathcal{A}}$, noted $(\sigma_{a,\,s})_{a \in \mathcal{A}}$, that were measured around $0.1 - 0.3$ across public series.

Action set

After running computationally intensive over test the competition's series, we derive an action set to train our bandit agent :

Action set

$\mathcal{A} = \{(count, 1), (count, 5), (countdual, 2), (binary, 3), (dual, 3)\}$

We use Thompson sampling select to train our agent online :

Thompson sampling

- Hypothesis $(\mathcal{P}_{a,s})_{a\in\mathcal{A}} \sim \mathcal{N}(\mu_{a,s}, \sigma = 0.2)$
- Initialize $(\hat{\mu_a}, \hat{\sigma_a})$ with (μ_0, σ_0)
- Draw samples $x_a \sim \mathcal{N}(\hat{\mu_a}, \hat{\sigma_a})$ for $a \in \mathcal{A}$
- Perform action $a^* = \underset{a}{arg\,min}\, x_a$ and observe reward $f_s(a_i)$
- Perform bayesian update on $\hat{\mu_a}$ and $\hat{\sigma_a}$

TS convergence on MIPcc23 series

$$CS = \frac{\sum_{i=1}^{50} \mu_{i,s}(a_i) - \mu_{i,s}(a_0)}{\sum_{i=1}^{50} \mu_{i,s}(a_s^*) - \mu_{i,s}(a_0)}$$

Series	Convergence score
bnd series 1	72%
bnd series 2	65%
obj series 1	75%
obj series 2	66%
rhs series 1	64%
rhs series 2	72%
rhs obj series 1	74%

Table: Convergence score of Thompson sampling on MIPcc23 public instances series.

Results

Series	Average $f_{s,\,i}$	Speed up
bnd series 1	0.992 ± 0.009	$-\mathbf{0.031 \pm 0.009}$
bnd series 2	0.881 ± 0.020	$-\mathbf{0.037 \pm 0.020}$
obj series 1	0.895 ± 0.006	$-\mathbf{0.022 \pm 0.006}$
obj series 2	0.891 ± 0.022	$-\mathbf{0.052 \pm 0.022}$
rhs series 1	0.875 ± 0.027	$-\mathbf{0.048 \pm 0.027}$
rhs series 2	1.004 ± 0.0001	0.001 ± 0.0001
rhs obj series 1	1.015 ± 0.006	-0.005 ± 0.006
mat series 1	1.050 ± 0.013	-0.009 ± 0.013
mat rhs bnd obj series 1	0.677 ± 0.021	$-\mathbf{0.061 \pm 0.021}$

Table: Averaged speed up obtained across public series. Results are averaged over 2,000 runs, with varying seed.

Conclusion

Criticism

- $\mathcal{A}$ build over public instances, no guarantee that any of the actions will be efficient on the hidden series
- Isn't it just a sophisticated but disguised approach to overfit the competition dataset ?

Strengths

- Sub-optimal actions also achieve significant speed up over public instance series
- We had only 50 instances to train our agent. State of the art methods obtain speed increases of only 4% while training over hundreds of instances.
- With larger dataset, more actions could be added to the action set, thus improving the power of generalization of our method.

Conclusion

Thanks for your attention !
Let’s hear from you :)