RÉSOLUTION DE PROBLÈMES COMBINATOIRES

CP-SAT

Loïc & Loïc

Table des matières

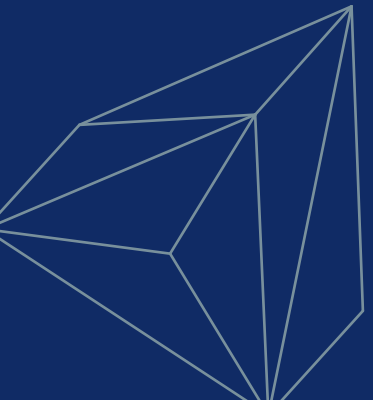
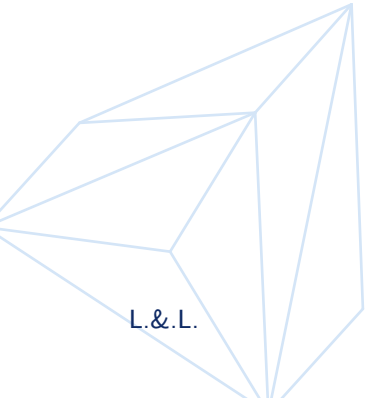
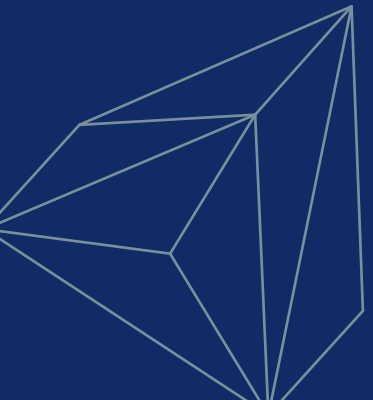


Table des matières

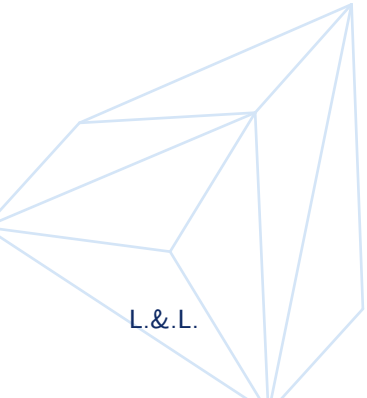


Historique



Nogoods, CDCL et Lazy Clause Generation

- Generalized nogoods in CSPs (Katsirelos et Bacchus 2005)
- GRASP-a new search algorithm for satisfiability (CDCL) (Silva et Sakallah 1996)
- Lazy clause generation: Combining the power of SAT and CP (and MIP?) solving (Stuckey 2010)
- Search is Dead, Peter J. Stuckey CP'13

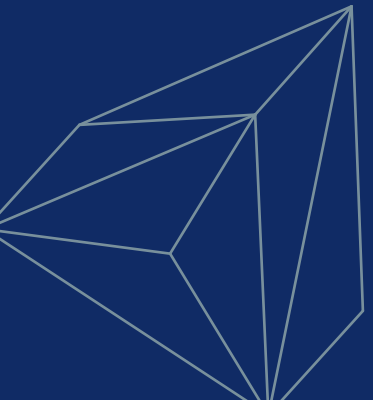


Evolutions au challenge MiniZinc

	Catégorie	2024	2023	2022	2021	2020	2019
Fixed	Gold	OR-Tools CP-SAT	OR-Tools	OR-Tools	OR-Tools	SICStus Prolog	OR-Tools
	Silver	Choco-solver CP-SAT	SICStus Prolog	SICStus Prolog	JaCoP	JaCoP	JaCoP
	Bronze	SICStus Prolog	Choco 4	JaCoP	PicatSAT	Choco 4	SICStus Prolog
Free	Gold	OR-Tools CP-SAT	OR-Tools	OR-Tools	OR-Tools	OR-Tools	OR-Tools
	Silver	PicatSAT	PicatSAT	PicatSAT	PicatSAT	PicatSAT	iZplus
	Bronze	iZplus	iZplus	Choco 4	SICStus Prolog	Mistral 2.0	PicatSAT
Parallel	Gold	OR-Tools CP-SAT	OR-Tools	OR-Tools	OR-Tools	OR-Tools	OR-Tools
	Silver	PicatSAT	PicatSAT	PicatSAT	PicatSAT	PicatSAT	iZplus
	Bronze	Choco-solver CP	Choco 4	Geas	iZplus + Choco 4	Mistral 2.0	Choco 4

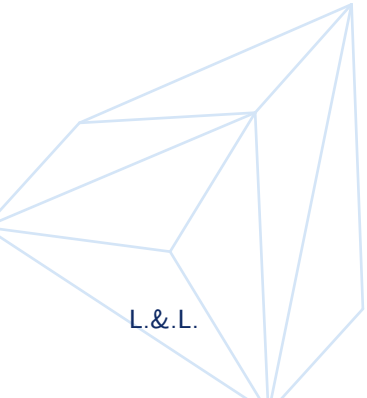
Tableau 1. – Les résultats des compétitions MiniZinc de 2019 à 2024 pour les catégories « Fixed », « Free » et « Parallel ».

CP-SAT : Qu'est ce à dire que ceci ?



Le Talk à l'origine de Or-Tools : « Search Is Dead »

- Comment fonctionnent les solveurs ?
- Combien de fois les techniques de recherche sont répétées ?
- Search is Dead



Comment fonctionnent les solveurs CP ?

- Algorithmes de propagation

Pour un problème (X, D, C), nous avons :

- un domaine D qui associe chaque variable x à ses valeurs possibles $D(x)$;
- pour chaque contrainte un (ou plusieurs) propagateurs $f_c : D \to D$
  - une fonction décroissante monotone (réduction de domaine)
  - supprime les valeurs qui ne font pas partie de la solution
- un solveur de propagation $D = \text{solv}(F, D)$
  - applique les propagateurs $f \in F$ – à plusieurs reprises – à D jusqu'à ce que $f(D) = D$ pour tout $f \in F$.

Problème de répétition : Combien de fois la recherche est-elle effectuée ?

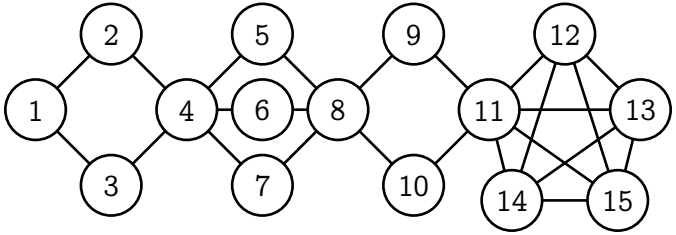


Fig. 1. – A 4 coulour graph problem

Problème de répétition : Combien de fois la recherche est-elle effectuée ?

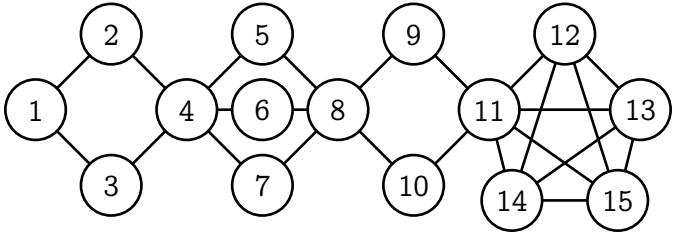


Fig. 2. – A 4 coulour graph problem

- Recherche dans l'ordre des noeuds : 462672 échecs

Problème de répétition : Combien de fois la recherche est-elle effectuée ?

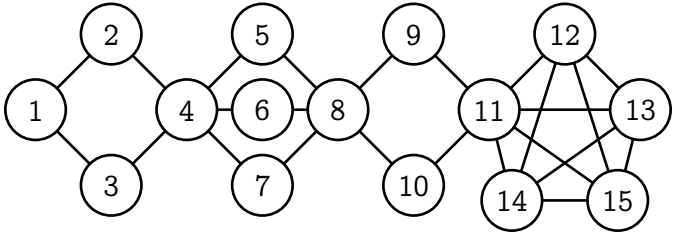


Fig. 3. – A 4 coulour graph problem

- Recherche dans l'ordre des noeuds : 462672 échecs, 18 avec apprentissage

Problème de répétition : Combien de fois la recherche est-elle effectuée ?

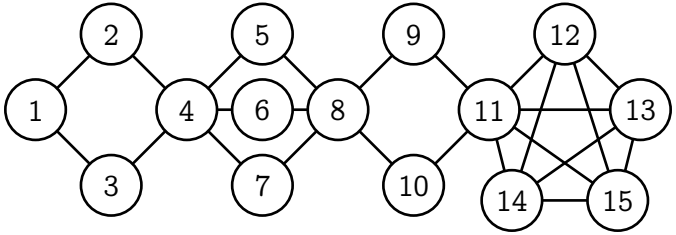


Fig. 4. – A 4 coulour graph problem

- Recherche dans l'ordre des noeuds : 462672 échecs, 18 avec apprentissage
- Suppression des symmétries : 19728 échecs

Problème de répétition : Combien de fois la recherche est-elle effectuée ?

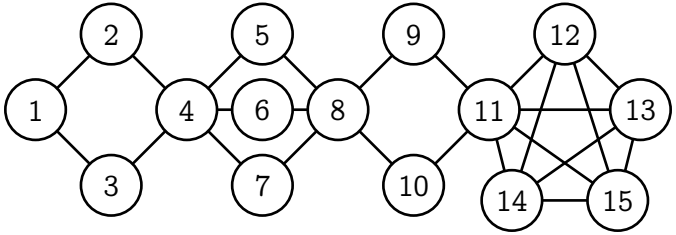


Fig. 5. – A 4 coulour graph problem

- Recherche dans l'ordre des noeuds : 462672 échecs, 18 avec apprentissage
- Suppression des symmétries : 19728 échecs, 19 avec apprentissage

Problème de répétition : Combien de fois la recherche est-elle effectuée ?

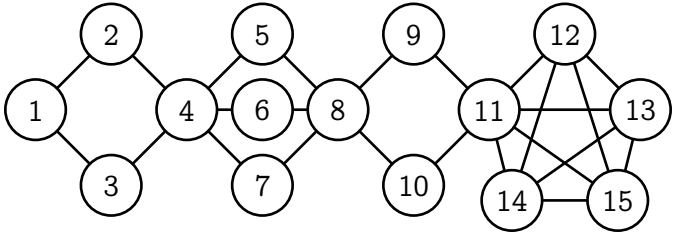


Fig. 6. – A 4 coulour graph problem

- Recherche dans l'ordre des noeuds : 462672 échecs, 18 avec apprentissage
- Suppression des symmétries : 19728 échecs, 19 avec apprentissage
- Inversion de l'ordre : 24 échecs

Problème de répétition : Combien de fois la recherche est-elle effectuée ?

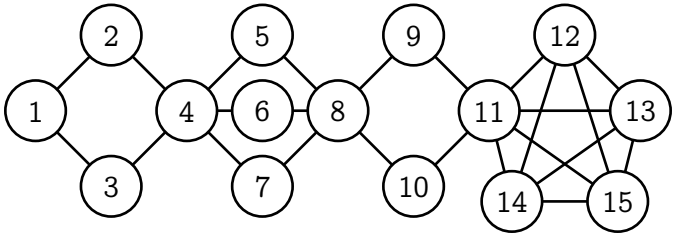


Fig. 7. – A 4 coulour graph problem

- Recherche dans l'ordre des noeuds : 462672 échecs, 18 avec apprentissage
- Suppression des symmétries : 19728 échecs, 19 avec apprentissage
- Inversion de l'ordre : 24 échecs, 18 avec apprentissage

Problème de répétition : Combien de fois la recherche est-elle effectuée ?

Résponse :

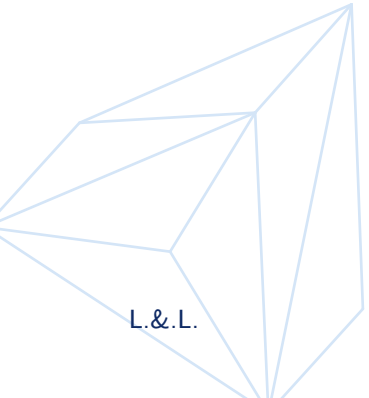
Beaucoup !

- Certaines méthodes permettent d'atténuer le problème :
  - Traitement de la symmétrie/dominance;
  - Restarts et stratégies de recherche dynamiques;
  - **Apprentissage** / Cache

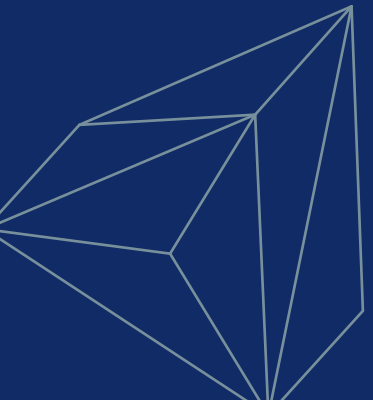
~~Rechercher~~ Prouver

```
include "alldifferent.mzn";

int: n;
array[1..n] of var 1..n: x;
constraint alldifferent(x);
constraint sum(x) < n * (n + 1) div 2;
```

```
include "alldifferent.mzn";

int: n;
array[1..n] of var 1..n: x;
array[1..n] of var 0..n * (n + 1) div 2: s;
constraint alldifferent(x);
constraint s[1] =
  x[1] /\ s[n] < n * (n + 1) div 2;
constraint forall(i in 2..n)(
    s[i] = x[i] + s[i - 1]
);
```



Et Or-Tools ?



Construction du Solver

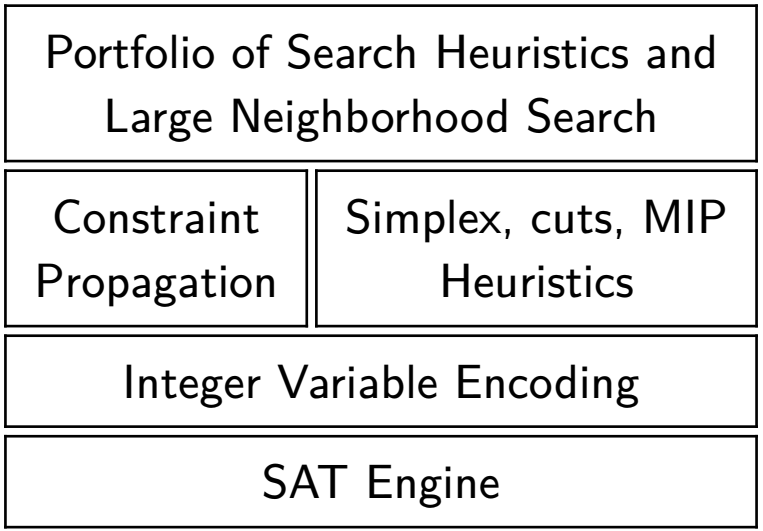


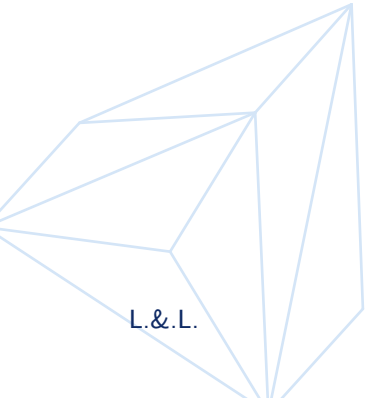
Tableau 2. – La construction du solveur Or-Tools

Différences entre CP → SAT et CP-SAT

- L'encodage des valeurs entières se fait avec deux booléens :

Pour une variable x comprise entre l et u, nous avons :

- un booléen pour $[x = v]$, pour d dans $[l; u - 1]$;
- un booléen pour $[x \leq v]$, pour d dans $[l; u]$.



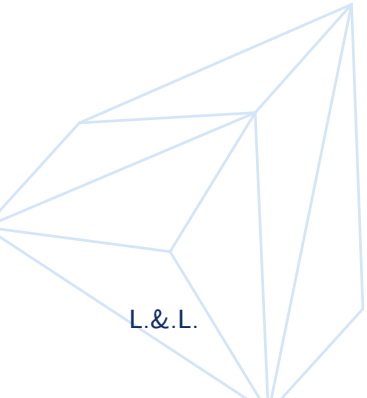
Différences entre CP → SAT et CP-SAT

- L'encodage des valeurs entières se fait avec deux booléens :

Pour une variable x comprise entre l et u, nous avons :

- un booléen pour $[x = v]$, pour d dans $[l; u - 1]$;
- un booléen pour $[x \leq v]$, pour d dans $[l; u]$.

L'encodage de valeur $(x = v)$ est statique.



Différences entre CP → SAT et CP-SAT

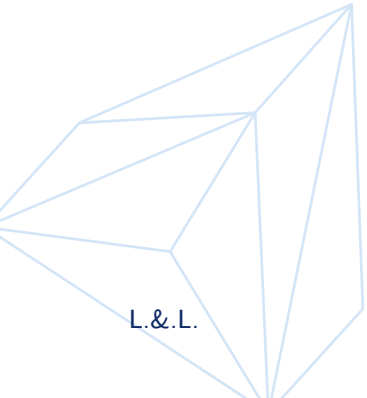
- L'encodage des valeurs entières se fait avec deux booléens :

Pour une variable x comprise entre l et u, nous avons :

- un booléen pour $[x = v]$, pour d dans $[l; u - 1]$;
- un booléen pour $[x \leq v]$, pour d dans $[l; u]$.

L'encodage de valeur $(x = v)$ est statique.

L'encodage d'ordre $(x \leq v)$ est dynamique.



Différences entre CP → SAT et CP-SAT

Restriction **aux changements atomiques** dans les domaines :

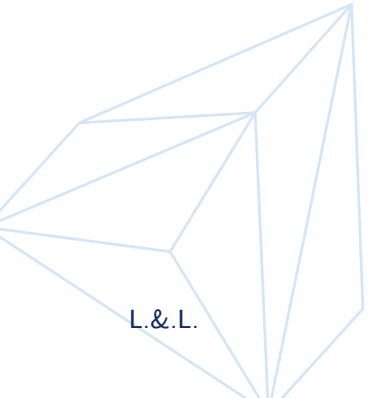
- $[x \leq d]$ (lui-même) ;
- $[x \geq d]$ $![x \leq d-1]$;
- $[x = d]$ (lui-même) ;
- $[x \neq d]$ $![x \neq d]$.

Clauses DOM pour modéliser la relation des booléens :

- $[x \leq d] \Rightarrow [x \leq d+1]$, pour d dans $[l; u-2]$;
- $[x = d] \Leftrightarrow [x \leq d] \wedge ![x \leq d-1]$, pour d dans $[l+1; i-1]$

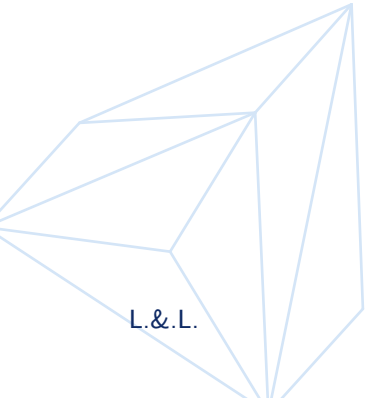
(I)LP

	LP	ILP
Domain	$\mathbb{R}$	$\mathbb{Z}$
Algorithms	Simplex Ellipsoids	Branch-and-cuts
Complexité	2^n, polynomal	NP



Réprésentation dans Or-Tools

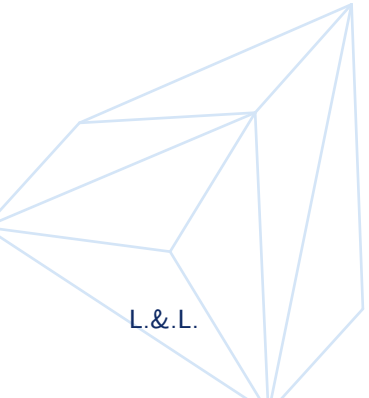
- Variables **entières** bornées : $x_i \in [lb_i, ub_i], x_i \in \mathbb{Z}$,
- Objectif linéaire et entier : $\text{minimize} \sum_i \text{obj}_i x_i, \text{obj}_i \in \mathbb{Z}$,
- Pool de contraintes linéaires : $[\text{lhs} \leq]\text{coeff}_i\text{obj}_i[\leq \text{rhs}], \text{coeff}_i \in \mathbb{Z}$.



Réprésentation dans Or-Tools

- Variables **entières** bornées : $x_i \in [lb_i, ub_i], x_i \in \mathbb{Z}$,
- Objectif linéaire et entier : minimize $\sum_i \text{obj}_i x_i, \text{obj}_i \in \mathbb{Z}$,
- Pool de contraintes linéaires : $[\text{lhs} \leq]\text{coeff}_i\text{obj}_i[\leq \text{rhs}], \text{coeff}_i \in \mathbb{Z}$.

Remarque ?

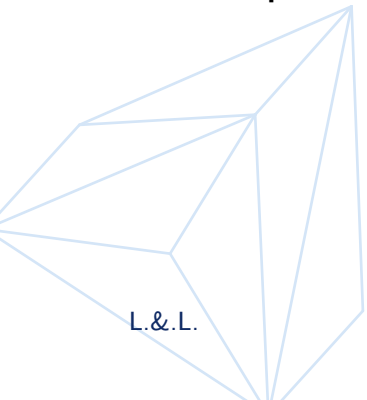


Réprésentation dans Or-Tools

- Variables **entières** bornées : $x_i \in [lb_i, ub_i], x_i \in \mathbb{Z}$,
- Objectif linéaire et entier : $\text{minimize} \sum_i \text{obj}_i x_i, \text{obj}_i \in \mathbb{Z}$,
- Pool de contraintes linéaires : $[\text{lhs} \leq]\text{coeff}_i\text{obj}_i[\leq \text{rhs}], \text{coeff}_i \in \mathbb{Z}$.

Comparaison avec les solveurs ILP

- Toutes les variables sont des entiers 64 bits bornés ;
- « Pas d'overflow »
- « Optional », toutes les contraintes sont déjà dans le moteur CP-SAT.



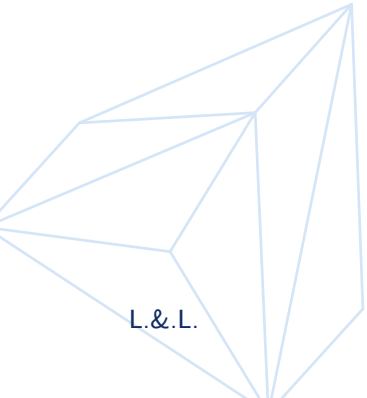
Qu'est ce qui est dans le modèle (I)LP ?

- Controllé par un paramètre `linearization_level` dans $\{0, 1, 2, 3\}$.
  0. Rien
  1. Constraintes linéaires ;
  2. Encodage des constraintes booléennes
     Encodage des variables entières
     Graphe d'implication
     At Most One
     etc.
  3. Constrainte circuit
     Produit d'inégalités
     Relaxation d'ordonancement.

Quand le modèle LP est utilisé ?

À chaque nœud ! Mais :

- Propagateur avec une basse priorité ;
- Effectue un nombre limité d'itération du simplex ;
- Commence avec un problème LP vide et ajoute des constraintes depuis le « pool » par petit paquets ;
- Supprime périodiquement des contraintes qui n'ont pas été utilisée les n dernières itérations.



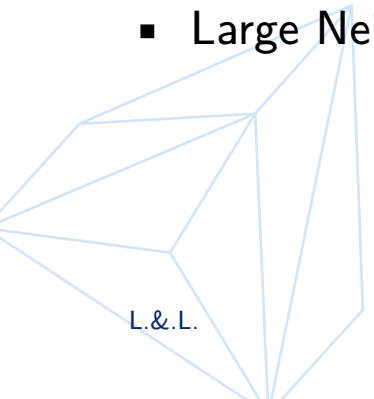
Pourquoi faire ?

Propagation de bornes

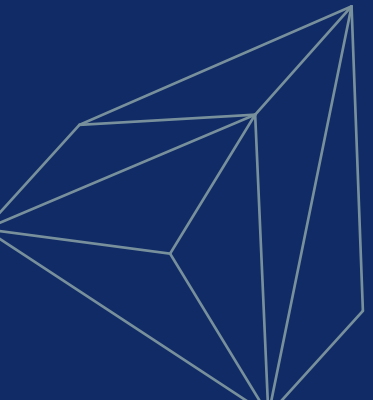
- Invalidité du modèle LP ;
- Borne min de l'objectif ;
- Borne min/max des variables.

Heuristiques

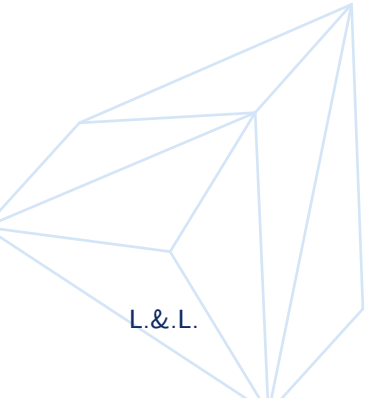
- Branchement : la valeur optimale LP peut être utilisée ;
- Large Neighborhood Search.



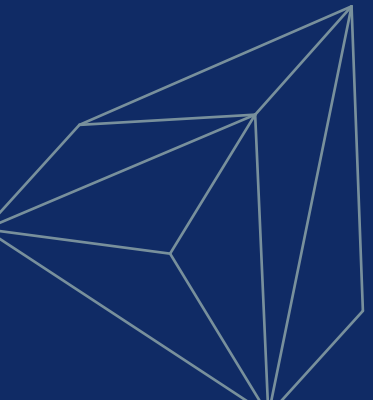
Conclusion



- Utiliser différentes techniques peut être très efficace ;
- La manière de modéliser les problèmes a un impact très important sur les performances ;
- Difficile de rattraper le travail effectué par les solveurs propriétaires historiques en MIP (CPLEX, Gurobi, CP Optimizer) ;
- Problématiques de parallélisation sur GPU (travail de NVIDIA en cours - LP uniquement pour le moment -).



Bibliographie



Katsirelos G, Bacchus F (2005) Generalized nogoods in CSPs. In: AAAI. p 390-396

Silva J M, Sakallah K A (1996) GRASP-a new search algorithm for satisfiability. In: Proceedings of International Conference on Computer Aided Design. p 220-227

Stuckey P J (2010) Lazy clause generation: Combining the power of SAT and CP (and MIP?) solving. In: International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming. p 5-9