LocalSolver

Black-box local-search for combinatorial optimization

www.localsolver.com

BOUYGUES

Aix*Marseille université

Thierry Benoist

Frédéric Gardi

Romain Megel

Bertrand Estellon

Karim Nouioua

LocalSolver

Plan

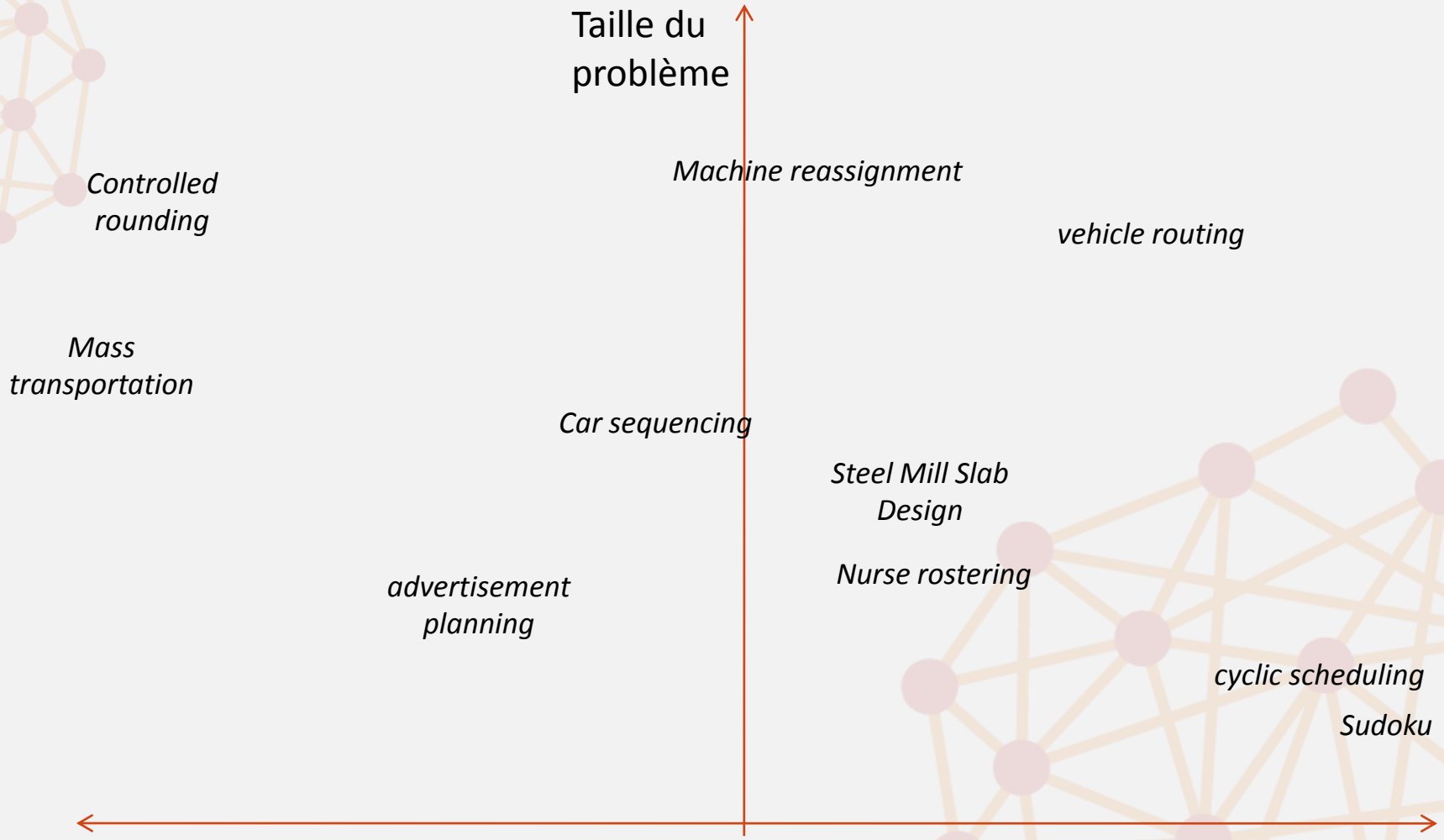
- Positionnement et « vision »
- Principes de base
- Un exemple détaillé *(car sequencing)*
- Exemples et applications
- Conclusion

Constat

- Pourquoi la Programmation Linéaire en Nombres Entiers (PLNE) est si populaire ?
  - Formalisme simple et générique
  - Simple d'utilisation: *model & run*

(La Programmation par Contraintes (PPC) suit cette voie)

Positionnement

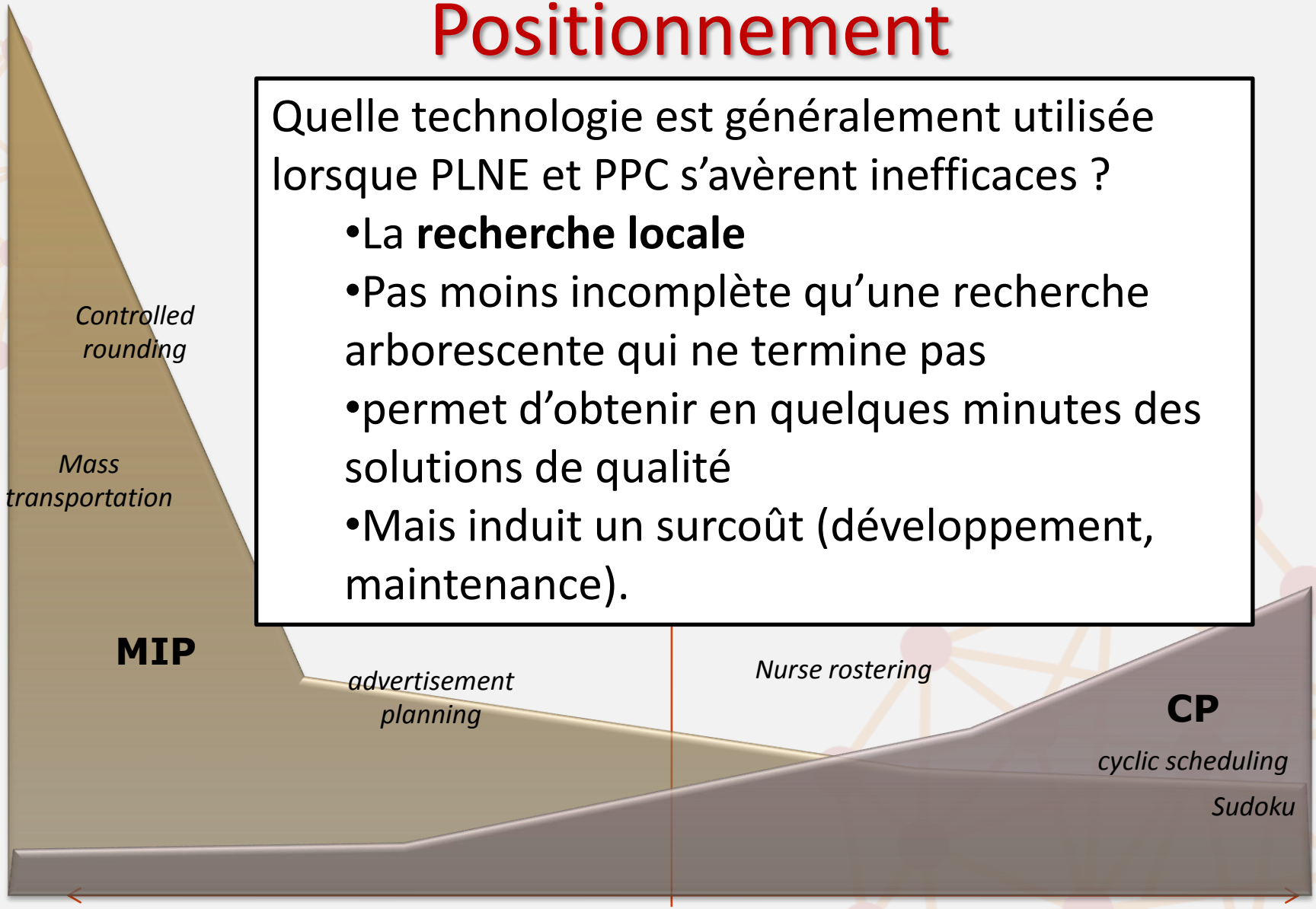


LocalSolver

Positionnement

Quelle technologie est généralement utilisée lorsque PLNE et PPC s'avèrent inefficaces ?

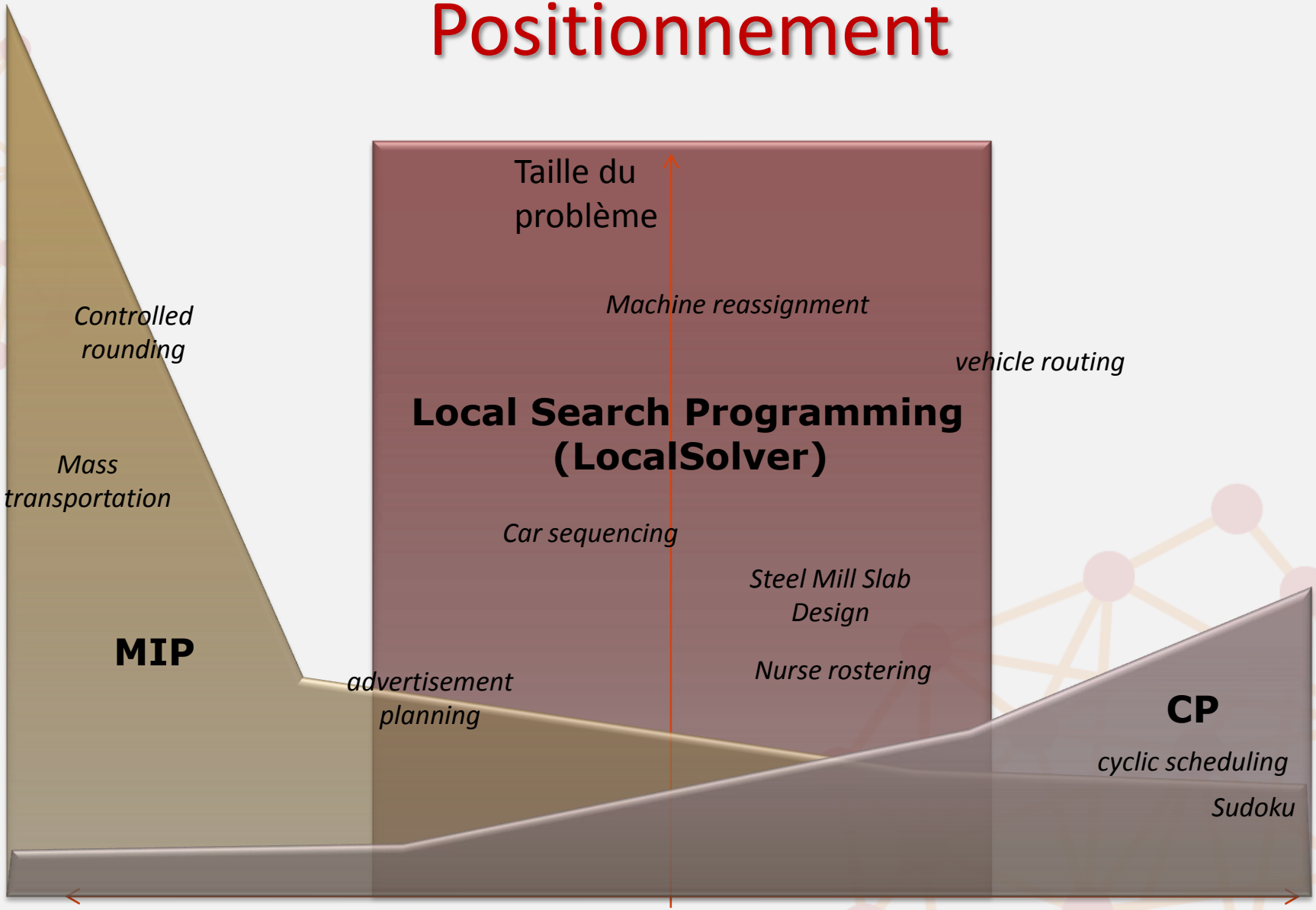
- La **recherche locale**
- Pas moins incomplète qu'une recherche arborescente qui ne termine pas
- permet d'obtenir en quelques minutes des solutions de qualité
- Mais induit un surcoût (développement, maintenance).



Problèmes à forte structure linéaire sous jacente

Problèmes purement combinatoires
(voire pure satisfaction)

Positionnement



Problèmes à forte structure linéaire sous jacente

Problèmes purement combinatoires
(voire pure satisfaction)

LocalSolver

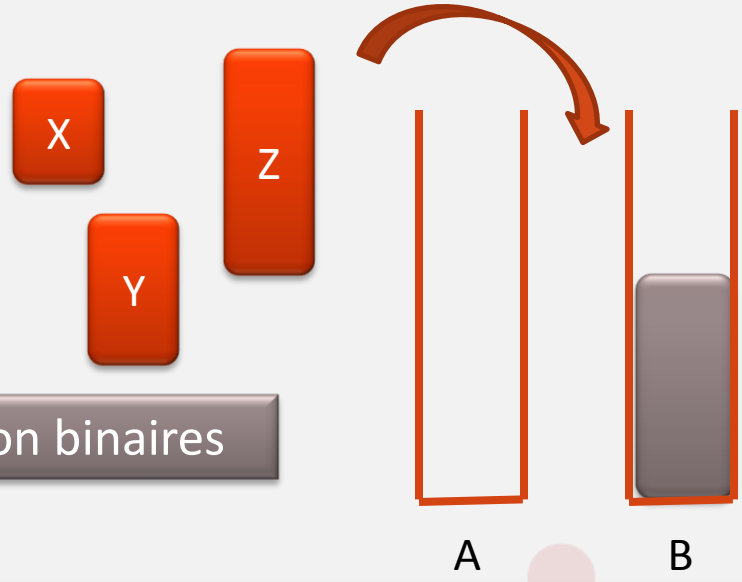
LocalSolver

- LocalSolver est un solveur boîte noire à base de recherche locale.
- Boîte noire =
  - l'utilisateur modélise le problème dans un formalisme simple et générique (*model*) et n'a pas à programmer de mouvement ou de recherche
- À base de recherche locale =
  - LocalSolver exploite efficacement ce formalisme pour appliquer des mouvements et une stratégie de recherche générique, mimant ce qu'un ingénieur aurait programmé à la main

PRINCIPES

Formalisme de modélisation

Un petit problème de *bin-packing* écrit au format LSP : 3 objets *x*, *y*, *z* à empiler dans 2 boîtes *A*, *B* de façon à minimiser la taille de la pile la plus grande, B contenant déjà un objet de taille 5.



Variables de décision binaires

Opérateurs arithmétiques *(sum, product, <, > , min,…)* et logiques (*and/or/if*)
→ ***Fortement non linéaire***

```
xA <- bool(); yA <- bool(); zA <- bool();
xB <- bool(); yB <- bool(); zB <- bool();
constraint  xA + xB == 1;
constraint  yA + yB == 1;
constraint  zA + zB == 1;
tailleA <- 2 * xA + 3 * yA + 4 * zA;
tailleB <- 2 * xB + 3 * yB + 4 * zB + 5;
minimize max(tailleA,tailleB);
maximize min(tailleA,tailleB);
```

Variables intermédiaires entières ou binaires

Objectifs lexicographiques

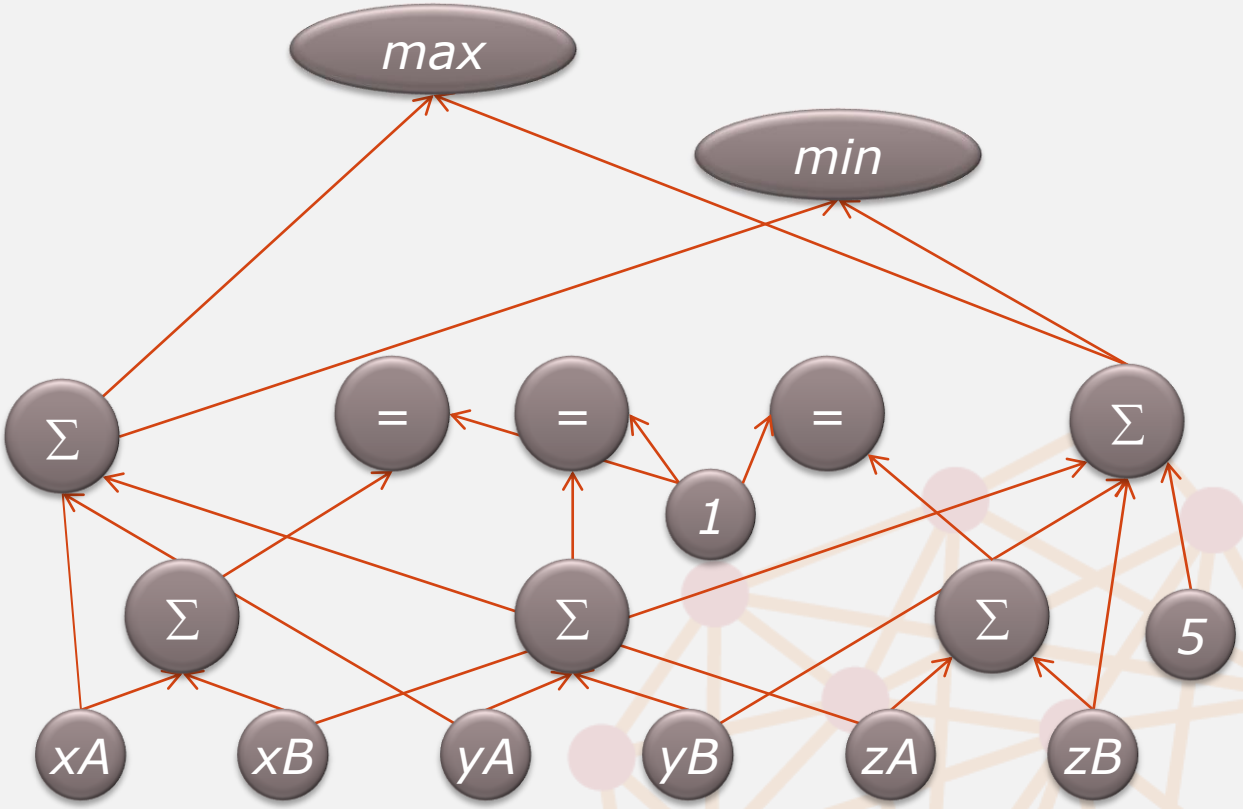
L'utilisateur n'a rien d'autre à faire que d'écrire ce modèle (approche déclarative = model & run)

Opérateurs disponibles

- Arithmétiques:
  - sum, product, min, max, abs, sqrt, square, divide, modulo
- Logiques
  - If, and, or, xor, not

Technologie

- En interne le modèle est représenté comme un treillis



Chaque opérateur est capable de se mettre à jour incrémentalement lorsque ses opérandes changent de valeur

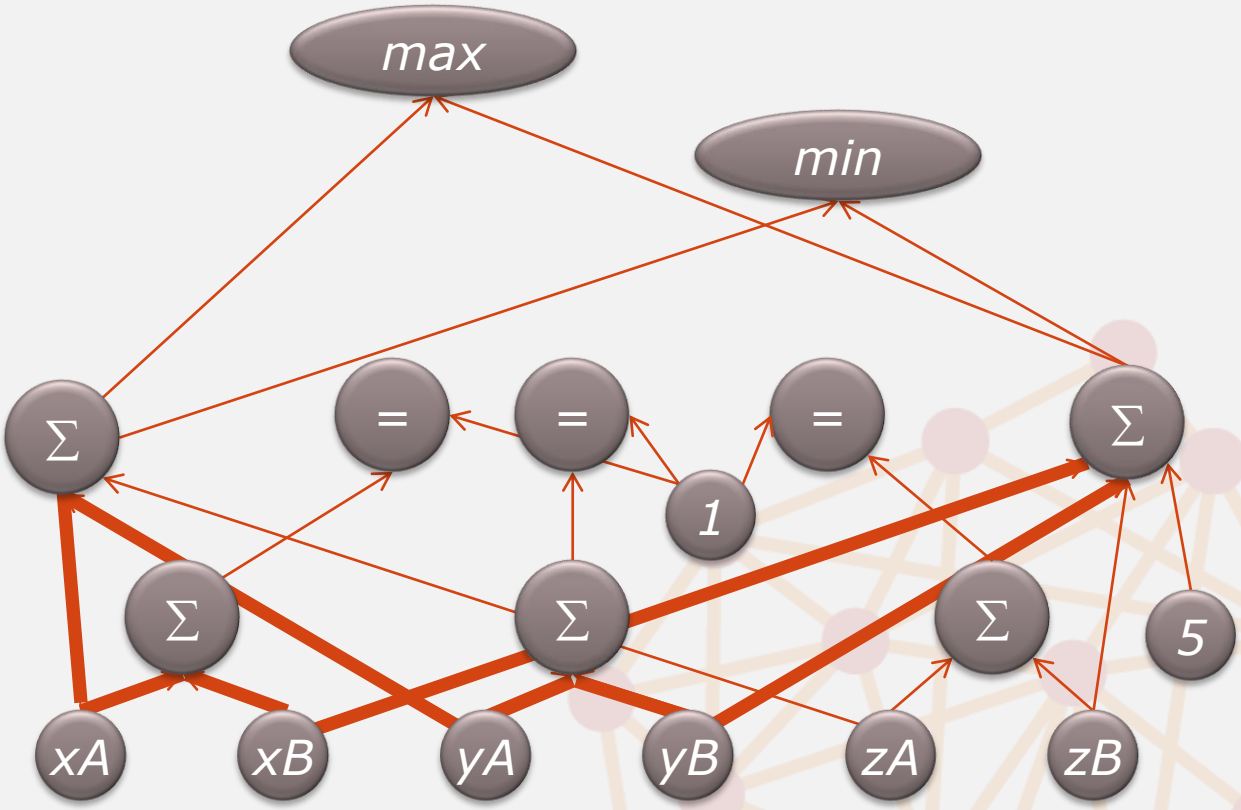
Un mouvement = un ensemble de variable binaires modifiées
L'évaluer = propager son impact sur l'objectif et les contraintes

Grâce à ces mécanismes incrémentaux,
LocalSolver visite des millions de solutions en quelques minutes

Technologie

- Le moteur utilise la structure de ce treillis pour effectuer des mouvements pertinents

Exemple: pour échanger la position des objets X et Y



En utilisant ces structures, LocalSolver effectue des mouvements proches de ce qu'un praticien expert aurait programmé

Pour en savoir plus

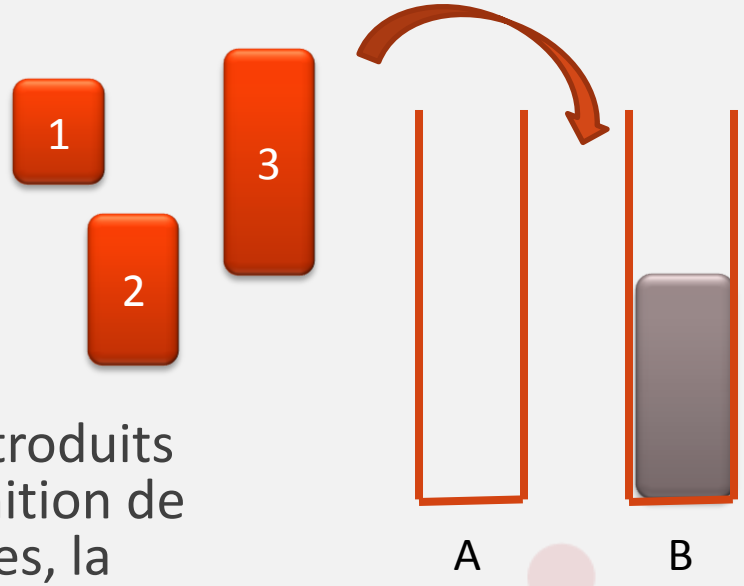
Sur la technologie et les performances :

T. Benoist, B. Estellon, F. Gardi, R. Megel, K. Nouioua.
LocalSolver 1.x: a black-box local-search solver for 0-1 programming.
4OR, A Quarterly Journal of Operations Research 9(3), pp. 299-316.

Langage de modélisation

```
A[i in 1..3] <- bool();
B[i in 1..3] <- bool();
For[i in 1..3] constraint A[i]+B[i]==1;
tailleA <-sum[i in 1..3](w[i] * A[i]);
tailleB <-sum[i in 1..3](w[i] * B[i])+5;
minimize max(tailleA,tailleB);
maximize min(tailleA,tailleB);
```

LSP



- En plus des opérateurs de modélisation introduits plus haut, la syntaxe autorise aussi la définition de boucles, de fonctions, de tables de variables, la lecture de données dans un fichier, etc.
- Ce langage est **adapté** à la recherche locale et moins **verbeux** que les modelers existants (OPL, Mosel, AMPL)
  - Interprété
  - Typage fort et dynamique (comme Python)
  - Déclaration implicite des variables (comme PHP, Lua)
  - Même opérateurs pour la modélisation et la programmation

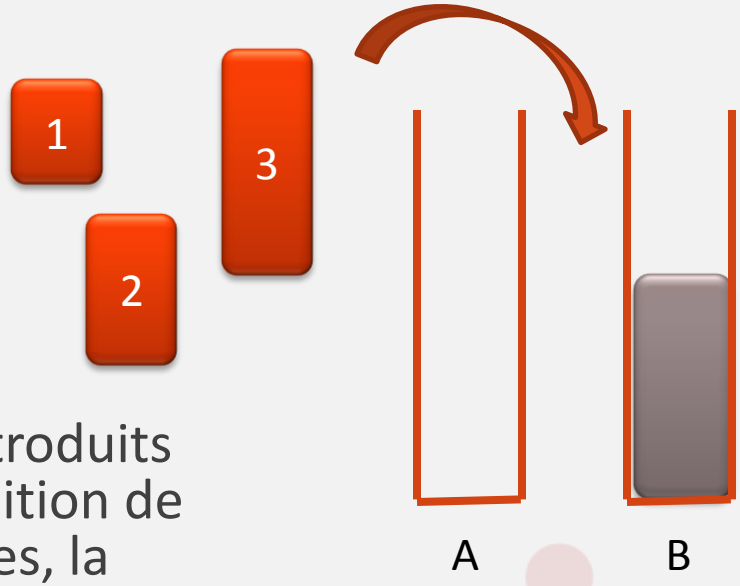
`A[3] = 8;`
➔ A est une table et contient 8 en position 3

`X * Y`
➔ Est une variable si X ou Y sont des variables, un entier sinon

Langage de modélisation

```
A[i in 1..3] <- bool();
B[i in 1..3] <- bool();
For[i in 1..3] constraint A[i]+B[i]==1;
tailleA <-sum[i in 1..3](w[i] * A[i]);
tailleB <-sum[i in 1..3](w[i] * B[i])+5;
minimize max(tailleA,tailleB);
maximize min(tailleA,tailleB);
```

LSP



- En plus des opérateurs de modélisation introduits plus haut, la syntaxe autorise aussi la définition de boucles, de fonctions, de tables de variables, la lecture de données dans un fichier, etc.

Le modeler permet de prototyper très rapidement.
Pour un déploiement en production LocalSolver offre des API C++, C#, Java

Distribution

LSP

```
A[i in 1..3] <- bool();
B[i in 1..3] <- bool();
```

C++

```
For (int i = 1; i <= 3; i++)
  For (int j = 1; j <= 2; j++)
```

Java

```
For (int i = 1; i <= 3; i++)
  For (int j = 1; j <= 2; j++)
```

C#

```
For (int i = 1; i <= 3; i++)
  For (int j = 1; j <= 2; j++)
    X[i][j] = m.createOperator(O_Bool);
For (int i = 1; i <= 3; i++) {
    var sum = m.createOperator(O_Sum,X[i]);
    m.addConstraint(m.createOperator(O_Eq,sum,1);
}
For (int j = 1; j <= 2; j++) {
  taille[j] = m.createOperator(O_Max);
  For (int i = 1; i <= 3; i++) {
    taille[j].addOperand(m.createOperator(O_Prod,w[i],X[i][j])
  }
}
m.addObjective(taille[1],OD_minimize);
m.addObjective(taille[2],OD_maximize);
```

Le modeler permet de prototyper très rapidement.
Pour un déploiement en production LocalSolver offre des API
C++, C#, Java

EXEMPLES

Car Sequencing

- Ordonnancement de véhicules sur une chaîne de production
- Objectif = espacer les options
  - ex: pas plus de 2 toits ouvrants toutes les 5 voitures («P/Q»)
  - mesure: dans chaque fenêtre de 5, pénalité basée sur les dépassements = max(n-2,0) avec *n* le nombre de toits ouvrants.
- Une classe est un ensemble de véhicules identiques
  - Ici avec 3 options A, B et C: AB est la classes des véhicules ayant les options A et B

AB	A	B	AB	A	ABC	C	A	B	C

Lecture des données

```
datafile = fileOpenRead(datafilename);
nbPositions = fileNextInt(datafile);
nbOptions = fileNextInt(datafile);
nbClasses = fileNextInt(datafile);
for[o in 1..nbOptions] P[o] = fileNextInt(datafile);
for[o in 1..nbOptions] Q[o] = fileNextInt(datafile);
for[c in 1..nbClasses] {
   card[c] = fileNextInt(datafile);
   options[c][1..nbOptions] = (fileNextInt(datafile)==1);
}
```

Typage fort (fileOpenRead renvoie un FILE)

Déclaration implicite

Syntaxe compacte

Modèle

2x AB 3 x A 2 x B ABC 2 x C

$X_{cp} = 1 \Leftrightarrow$ le véhicule en position p est de la classe c

```
X[c in 1..nbClasses][p in 1..nbPositions] <- bool();

for[c in 1..nbClasses]
   constraint sum[p in 1..nbPositions](X[c][p]) == card[c];

for[p in 1..nbPositions]
   constraint sum[c in 1..nbClasses](X[c][p]) == 1;

op[o in 1..nbOptions][p in 1..nbPositions] <-
            or[c in 1..nbClasses : options[c][o]](X[c][p]);

nbVehicles[o in 1..nbOptions][j in 1..nbPositions-Q[o]+1] <-
          sum[k in 1..Q[o]](op[o][j+k-1]);

violations[o in 1..nbOptions][j in 1..nbPositions-Q[o]+1] <-
          max(nbVehicles[o][j] - P[o]), 0 );

obj<-sum[o in 1..nbOptions][p in 1..nbPositions-Q[o]+1](violations[o][p]);
minimize obj;
```

Et c’est tout !

Résolution

- Que va faire LocalSolver pour ce modèle ?
  1. Trouver une solution initiale (ici une affectation aléatoire des véhicules)
  2. Appliquer les mouvements génériques

Echanges à 2

Echanges à 3 ou plus

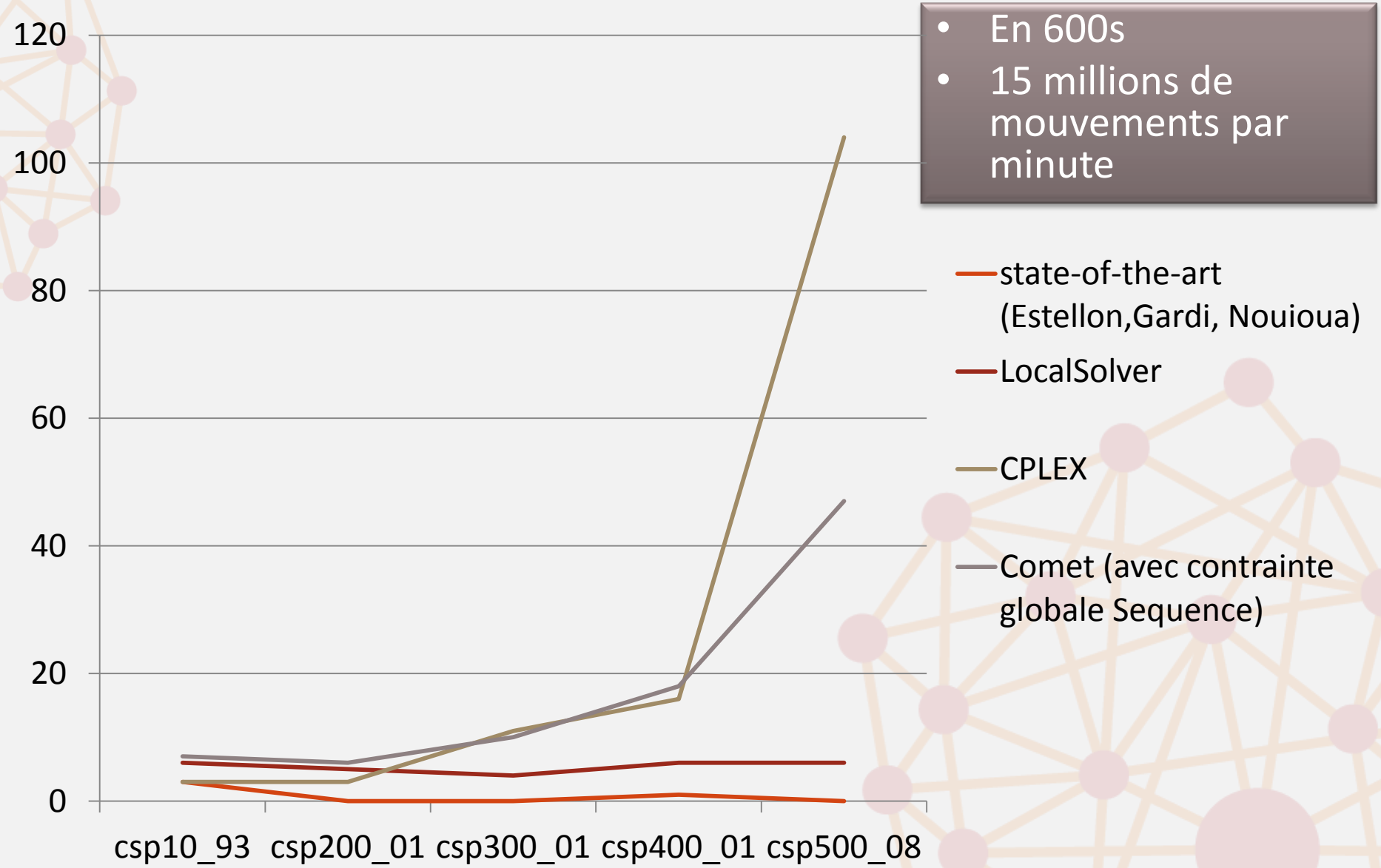
Déplacements simples

Contrainte violée !
(2 voitures à la même position)

- A noter:
  - Les déplacements simples seront éliminés après quelques secondes car ils échouent systématiquement.
  - La stratégie globale est un recuit simulé randomisé (paramétrable)
  - LocalSolver lance en parallèle plusieurs recherches (nombre de threads paramétrable)
  - Certains mouvements seront ciblés sur les fenêtres avec dépassement

Résultats

- En 600s
- 15 millions de mouvements par minute



Car Sequencing

Et s'il est interdit d'avoir plus de 10 voitures consécutives de la même couleur ?


```
Color[p in 1..nbPositions] <- sum[c in 1..nbClasses](Color[c] * X[c][p]);
Same[p in 2..nbPositions] <- color[p] == color[p-1];
TooLong[p in 11..nbPositions] <- and[j in p-10..p](Same[j]);
For p in 11..nbPositions constraint not(TooLong[p]);
```

Les variables de décision ne changent pas

Et si trouver une solution faisable n'est pas évident ?

```
Minimize sum[i in 11..nbPositions](TooLong[i]);
Minimize sum[i in 1..nbPositions](Penalty[i]);
```

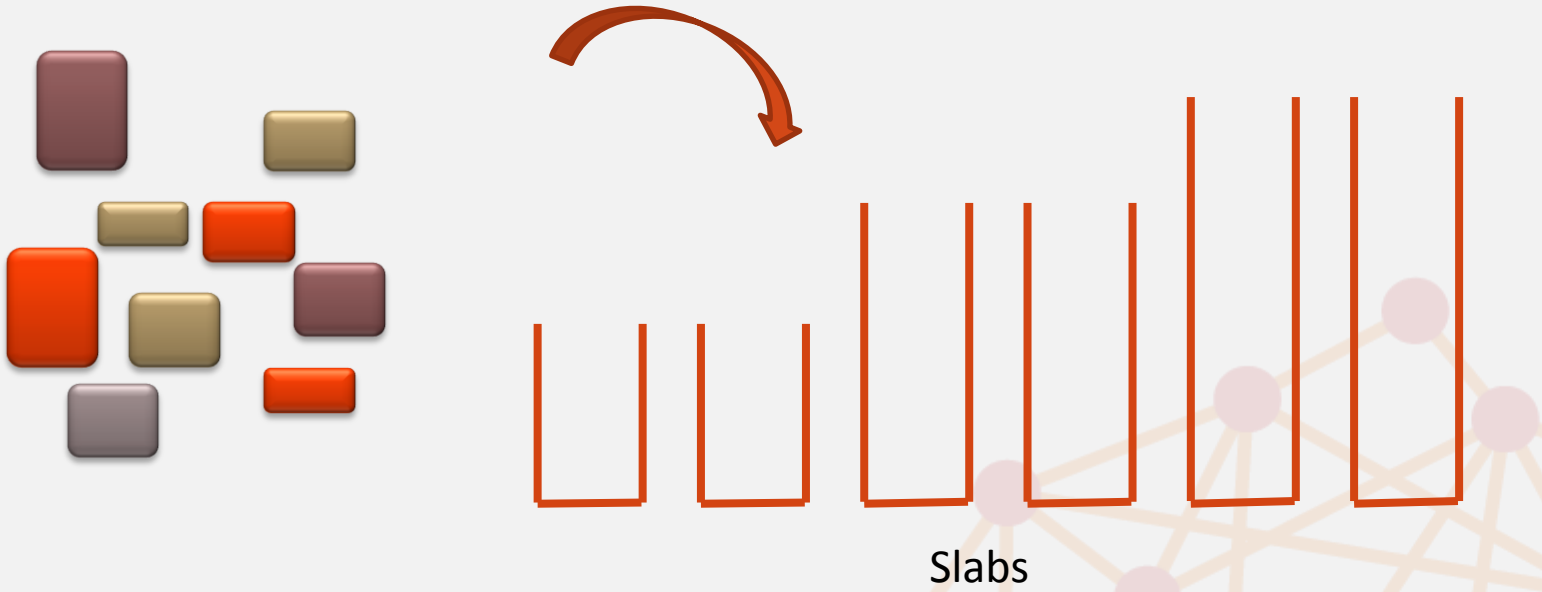
Et pour le challenge roadef 2005:

```
Maximize sum[i in 2..nbPositions](Same[i]);
Minimize sum[i in 1..nbPositions](Penalty[i]);
```

- Sur les instances du challenge Roadef 2005 (Renault)
  - LocalSolver se classe entre la 16eme et la 18eme position (sur 19 candidats)
- Avec un programme de 120 lignes !

Steel Mill Slab Design

- « Bin-packing » industriel
- Affectation de commandes d'acier à des « slabs » de capacités prédéfinies



Modèle

$X_{ij} = 1 \Leftrightarrow$ La commande i est affectée au slab j

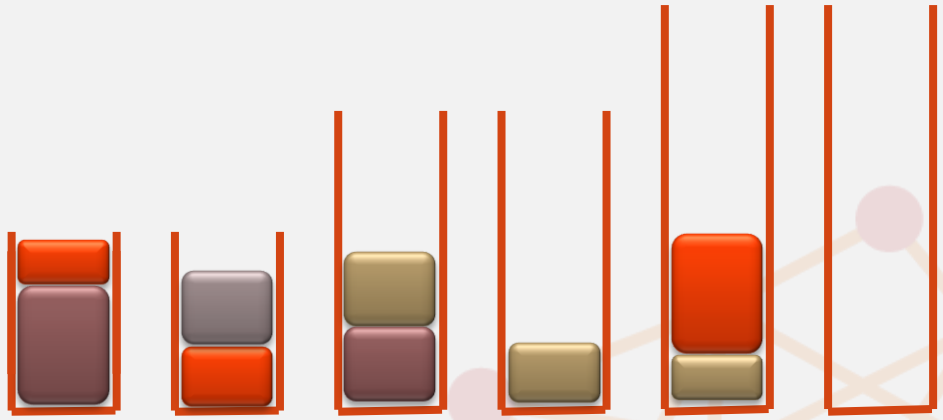
Steel Mill Slab Design

- « Bin-packing » industriel
- Affectation de commandes d'acier à des « slabs » de capacités prédéfinies

Déplacement

Echange

Chaîne d'éjection

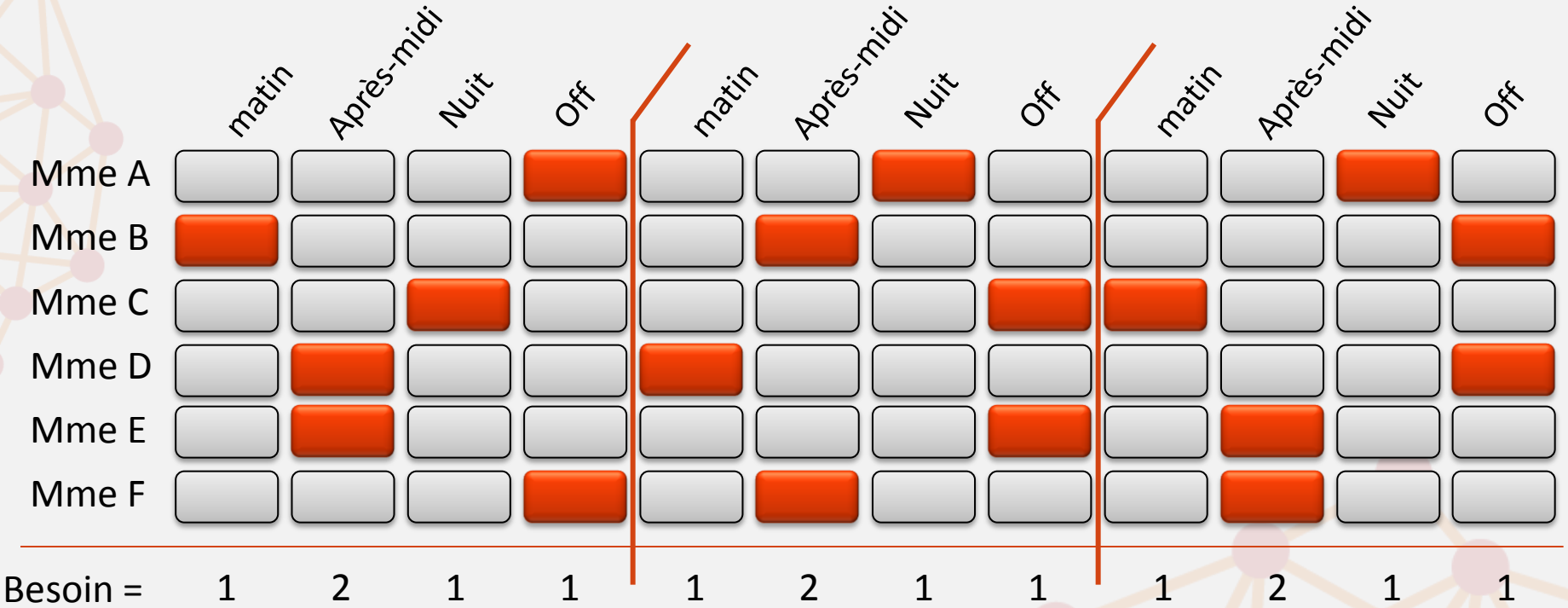


Dans un bon modèle:

• une valeur qui peut se déduire d'une ou plusieurs autres est définie par l'opérateur <- (c'est une variable **intermédiaire**)

• il suffit de modifier un petit nombre de variables de **décision** pour passer d'une solution faisable à une autre

Nurse rostering

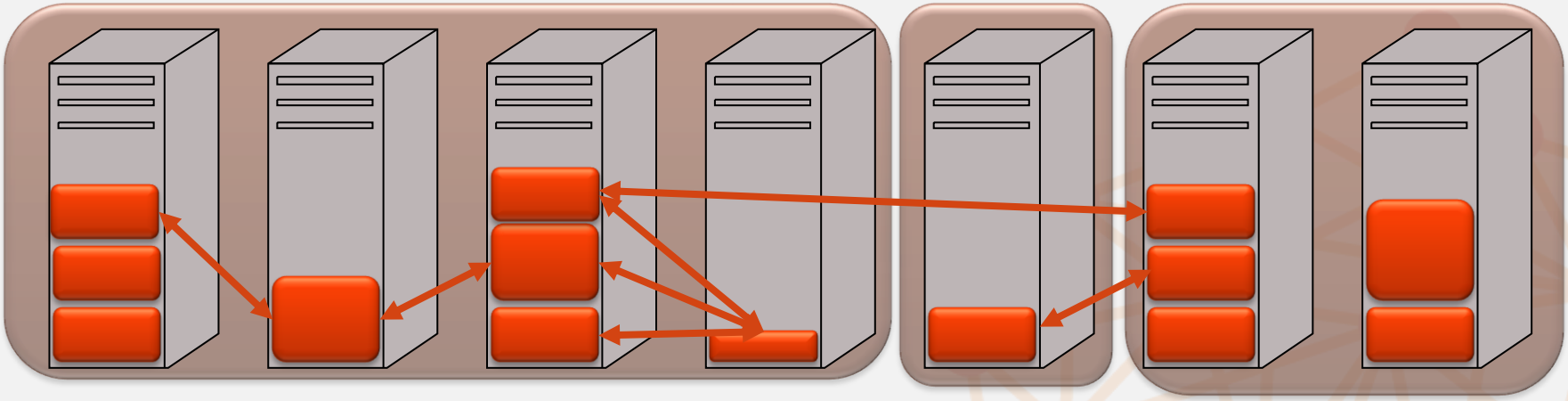


- Préférences:
  - Jours consécutifs de travail ou de repos
  - Enchaînements interdits de shifts
  - Shifts identiques samedi et dimanche pour les week-end travaillés
  - Repos minimums

Les opérateurs disponibles (arithmétiques et logiques) permettent de modéliser des relations fortement non linéaires

Challenge Roadef 2012

- Des processus sur des machines, dépendants les uns des autres, à réaffecter pour optimiser une fonction de coût



Plus de 100 000 variables binaires

Temps de développement: 4 heures

APPLICATIONS

Affectation des emplacements préférentiels

- Problème d'affectation avec contraintes non linéaires (max)
- 5 objectifs lexicographiques
- 10 000 variables binaires

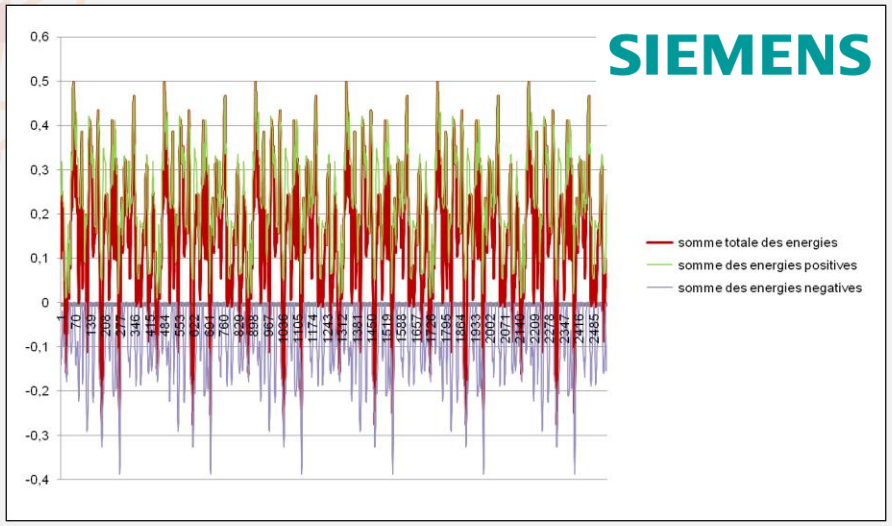
- PLNE: 6 jours de travail (modèle + décomposition objectifs)
- Recherche locale spécifique : 18 jours de travail (algorithme)
- LocalSolver : 3 jours de travail (modèle)

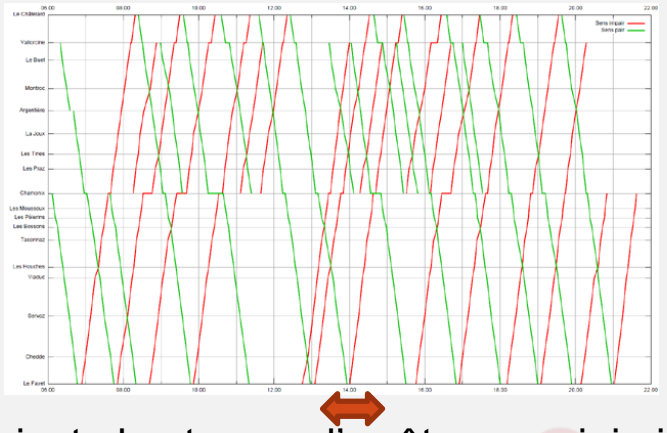
Solutions quasi optimales en moins de 60 secondes
4 millions de mouvements (= solutions visitées) par minute

LocalSolver en exploitation depuis plus d'un an

Optimisation énergétique d'une ligne de métro

EURODECISION
OPERATIONAL RESEARCH





On ajuste les temps d'arrêt pour minimiser l'énergie de freinage électrique perdue. 5000 variables binaires : une par date de départ possible de chaque gare (granularité en secondes).

Comparaison de CPLEX vs LocalSolver sur ce modèle (distance à l'optimum en %)

	LocalSolver 1.1	CPLEX 12.2
10 secondes	5 %	10 %
1 minute	1 %	4 %
20 minutes	0 %	1 %
2 heures	0 %	0 %

Feuille de route

- LocalSolver 2.0 en janvier 2012
- Développements futurs
  - Variables de décision entières
  - Structures ensemblistes
  - Couvrir l'ordonnancement et le routing
  - Variables de décision continues

Toujours en restant en mode « model & run »

Conclusion

- LocalSolver permet de traiter des problèmes fortement non linéaires et de grandes tailles, dans un mode « *model & run* »
- → une technologie très simple à essayer avant de se lancer dans le développement d'un algorithme dédié

LocalSolver

Black-box local-search for combinatorial optimization

www.localsolver.com