LocalSolver

LAAS CNRS

Réparation de solutions par propagation de réseaux d'inégalités dans LocalSolver

Léa Blaise

lblaise@localsolver.com

www.localsolver.com

LAAS-CNRS

ROADEF 2020

Montpellier

Présentation rapide du solveur LocalSolver

Solveur de type « model and run »

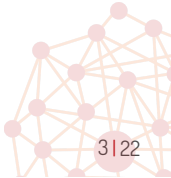
- Formalisme de modélisation mathématique simple
- Modélisation non linéaire et ensembliste
  - Variables de décision : bool, int, float, set, list
  - Grand nombre d'opérateurs logiques et arithmétiques :
    comparaisons, relations logiques, sum, prod, min, max, cos, exp, ...

Solveur global, combine des techniques exactes et heuristiques :

- Simplexe, points intérieurs, lagrangien augmenté, branch and bound, propagation...
- **Recherche locale**

Problématique

Difficultés de la recherche locale sur le Jobshop



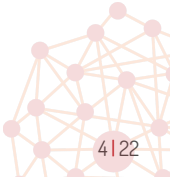
Jobshop

Données : m machines
n jobs composés chacun de m tâches (une par machine)
durées fixes

Contraintes de précédence : tâches de chaque job à réaliser dans un ordre donné

Contraintes de ressource disjonctive : une machine ne traite qu'une tâche à la fois

Objectif : minimiser le makespan



Jobshop – Modélisation avec LocalSolver

```
1  // décisions entières : dates de début des tâches
2  start[1..nbJobs][1..nbMachines] <- int(0, M);
3  end[j in 1..nbJobs][m in 1..nbMachines] <- start[j][m] + duration[j][m];
4
5  // contraintes de précédence
6  for [j in 1..nbJobs][k in 1..nbMachines-1]
7      constraint start[j][machine[j][k+1]] >= end[j][machine[j][k]];
8
9  // contraintes de ressource disjonctive : tâches prises deux à deux
10 for [m in 1..nbMachines][j1 in 1..nbJobs-1][j2 in j1+1..nbJobs]
11     constraint start[j1][m] >= end[j2][m] || start[j2][m] >= end[j1][m];
12
13 // makespan
14 minimize max[j in 1..nbJobs](end[j][machine[j][nbMachines]]);
```

Problème rencontré par LocalSolver sur le Jobshop

Mauvaises performances de LocalSolver 8.5 sur le Jobshop :

- Bonne solution $\Rightarrow$ contraintes de précédence et de ressource serrées
- Beaucoup de petits changements pour passer d'une solution de makespan x à une solution de makespan $x - 1$

Jobshop – Amélioration de 1 entre deux solutions réalisables

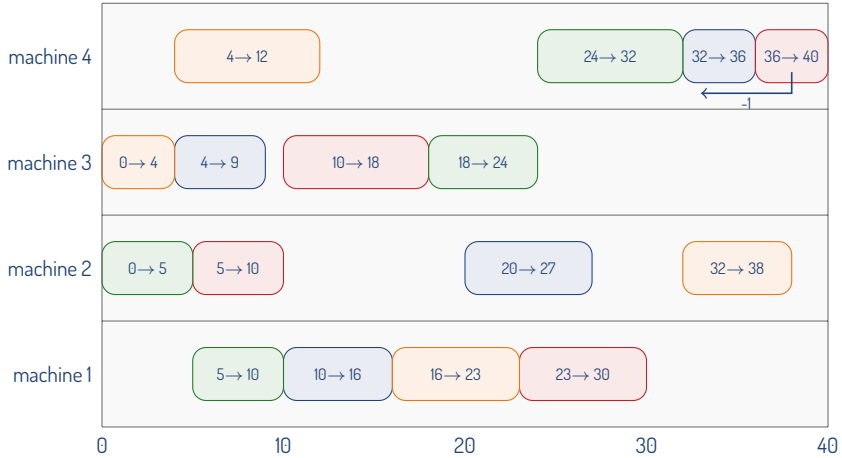


Figure – Solution de valeur 40 ⇒ décalage vers la gauche

Jobshop – Amélioration de 1 entre deux solutions réalisables

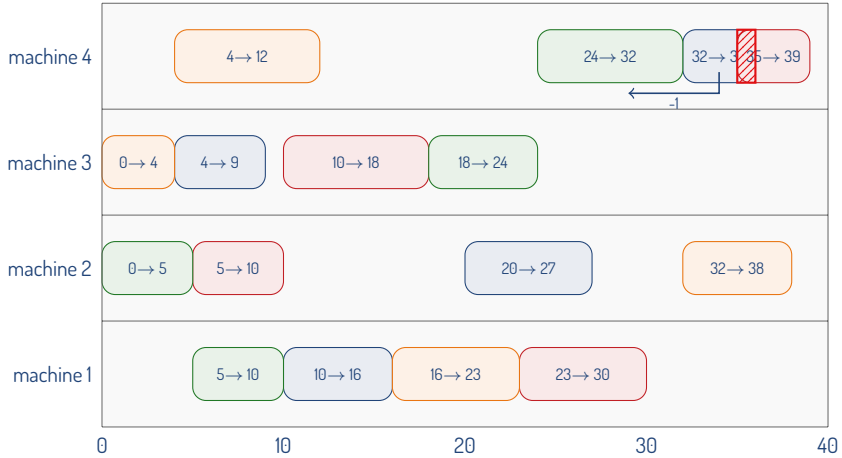


Figure – Conflit (ressource) ⇒ décalage vers la gauche

Jobshop – Amélioration de 1 entre deux solutions réalisables

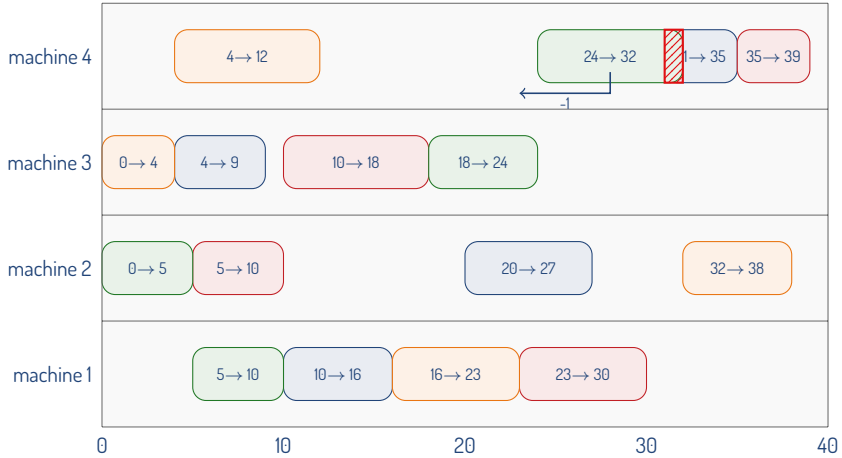


Figure – Conflit (ressource) ⇒ décalage vers la gauche

Jobshop – Amélioration de 1 entre deux solutions réalisables

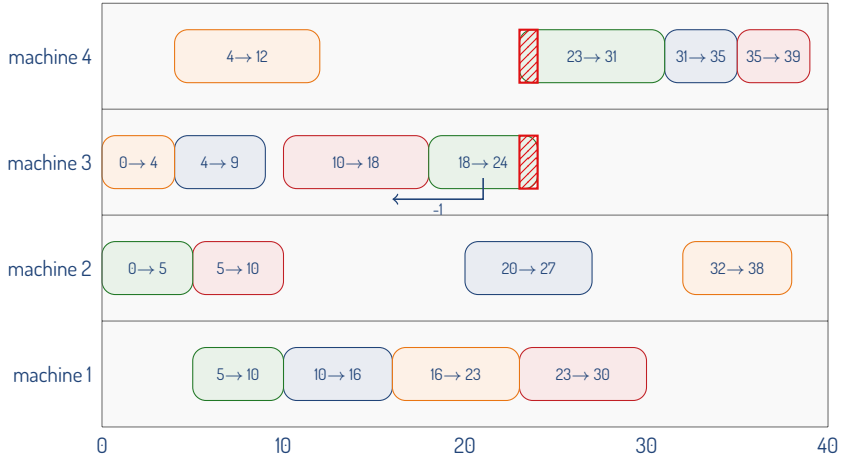


Figure – Conflit (précédence) ⇒ décalage vers la gauche

Jobshop – Amélioration de 1 entre deux solutions réalisables

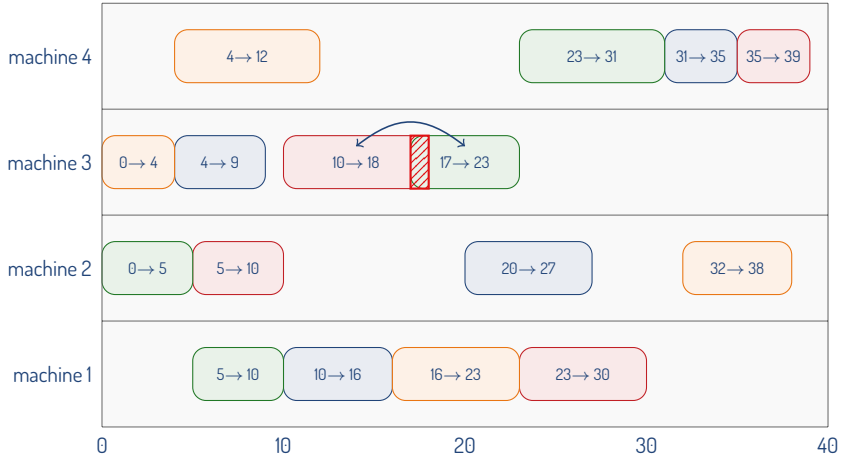


Figure – Conflit (ressource) ⇒ échange

Jobshop – Amélioration de 1 entre deux solutions réalisables

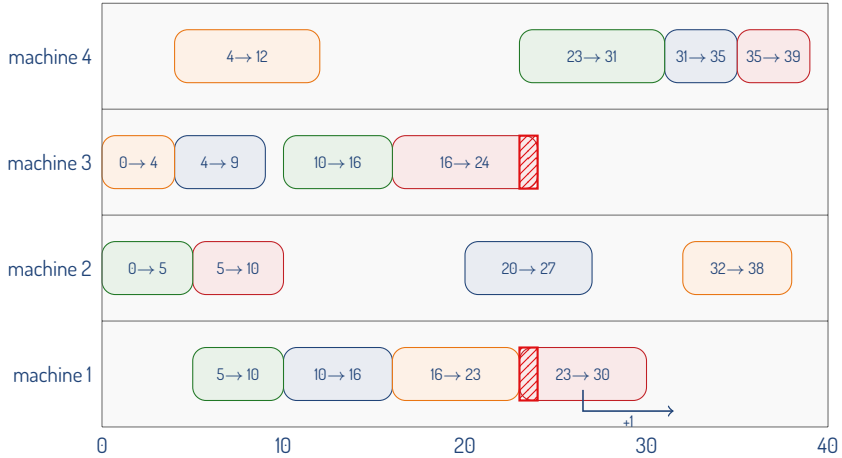


Figure – Conflit (précédence) ⇒ décalage vers la droite

Jobshop – Amélioration de 1 entre deux solutions réalisables

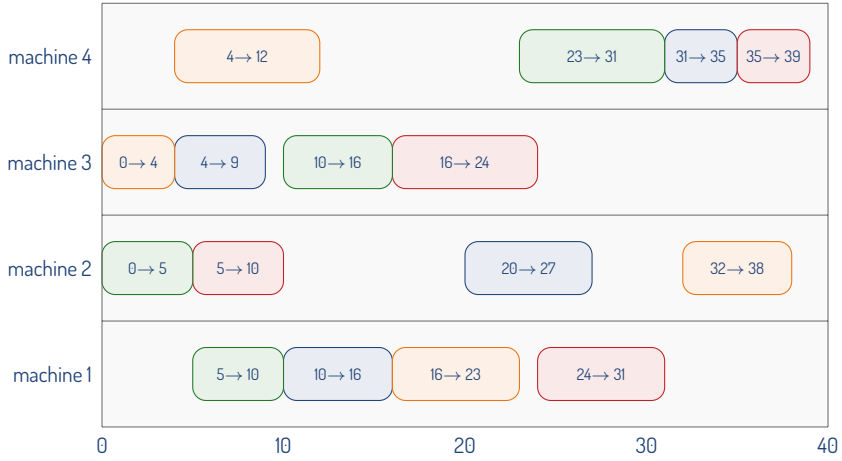


Figure – Solution de valeur 39

Problème rencontré par LocalSolver sur le Jobshop

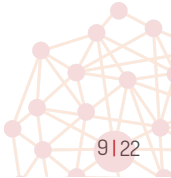
Passage d'une bonne solution à une autre :

- Changements de faible amplitude, mais sur un grand nombre de variables
- Mouvements de recherche locale sur les variables entières (dates de début) insuffisants

⇒ Il faut trouver une autre solution

Solution mise en place

Réparation par propagation des contraintes



Solution mise en place : réparation par propagation des contraintes

Détection dans le modèle de certaines contraintes de formes particulières :

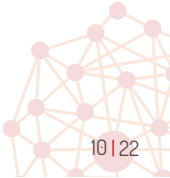
- Contraintes de précédence (généralisées) : $aX + bY \leq c$

```
start[j][m+1] >= end[j][m];
```

- Contraintes de ressource disjonctive (généralisées) : $\bigvee_i (a_i X_i + b_i Y_i \leq c_i)$

```
start[j1][m] >= end[j2][m] || start[j2][m] >= start[j1][m];
```

$\Rightarrow$ Réseau de contraintes, que l'on propage pour réparer une solution non réalisable



Principe de la propagation dans LocalSolver

« Propagation » pas exactement au sens de la CP

- Solution initiale réalisable
  Mouvement de recherche locale ⇒ solution non réalisable
- Pas de propagation des réductions de domaine
  Réparations successives des contraintes : en modifiant juste assez les variables
  Respect des choix précédents : modification des variables toujours dans le même sens
- Situation finale :
  Solution réalisable : on a réussi à réparer toutes les contraintes
  Échec de la propagation ⇒ on annule tout pour revenir à la solution réalisable de départ

Objectif : trouver une solution réalisable en étendant un mouvement non réalisable

Réparation d'une contrainte de précédence : déterministe

Réparation d'une contrainte de précédence ($aX + bY \leq c$) :

- On a cassé la contrainte en bougeant une des variables
- Pas d'autre choix que de réparer avec l'autre variable

Exemple : $start[t] \geq end[t']$

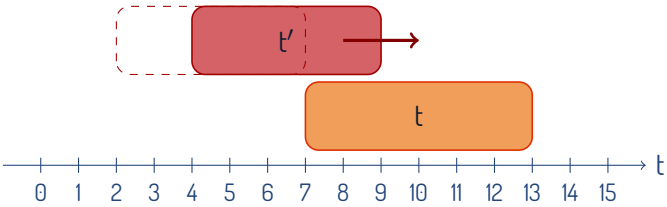


Figure – Une seule façon de réparer : décaler la tâche t vers la droite

Réparation d'une contrainte de précédence : déterministe

Réparation d'une contrainte de précédence ($aX + bY \leq c$) violée :

- Au plus une des variables (X) peut bouger dans le sens de la réparation
- On modifie X juste assez pour réparer la contrainte : $X \leftarrow \frac{c-bY}{a}$

Équivalent à une propagation particulière des bornes :

- Réduction de domaine propagée seulement si le support est exclu
- Les modifications se font toujours dans le même sens
  $\Rightarrow$ Modifie une seule borne et toujours la même

Réparation d'une contrainte de ressource disjonctive : non déterministe

Réparation d'une précédence dans une disjonction :

- Une autre précédence de la disjonction peut être devenue vraie
- On ne répare pas forcément la précédence qui était vérifiée initialement

Exemple : $(\text{start}[t'] \geq \text{end}[t]) \;||\; (\text{start}[t] \geq \text{end}[t'])$

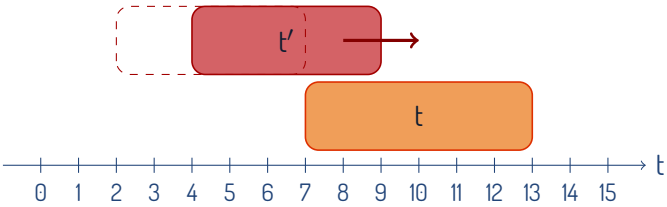


Figure – Plusieurs façons de réparer

Réparation d'une contrainte de ressource disjonctive : non déterministe

Réparation d'une disjonction de précédences $\bigvee_i (a_i X_i + b_i Y_i \leq c_i)$:

- On choisit une précédence au hasard dans la disjonction et on la répare

Réparation de la précédence $(aX + bY \leq c)$ choisie dans la disjonction :

- Si seul X peut bouger dans le sens de la réparation :
  On répare comme une contrainte de précédence
- Si X et Y peuvent bouger dans le sens de la réparation :
  On choisit au hasard une façon de réparer

Réparation d'une contrainte de ressource disjonctive : non déterministe

Réparation (non déterministe) d'une précédence de la disjonction :

$$(\mathbf{start}[\mathbf{t'}] \geq \mathbf{end}[\mathbf{t}]) \;||\; (\mathrm{start}[t] \geq \mathrm{end}[t'])$$

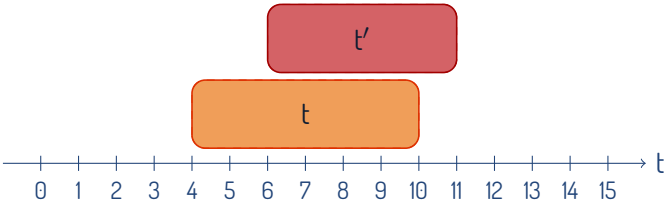


Figure – Contrainte non respectée

Réparation d'une contrainte de ressource disjonctive : non déterministe

Réparation (non déterministe) d'une précédence de la disjonction :

$$(\mathbf{start}[\mathbf{t'}] \geq \mathbf{end}[\mathbf{t}]) \;||\; (\mathrm{start}[t] \geq \mathrm{end}[t'])$$

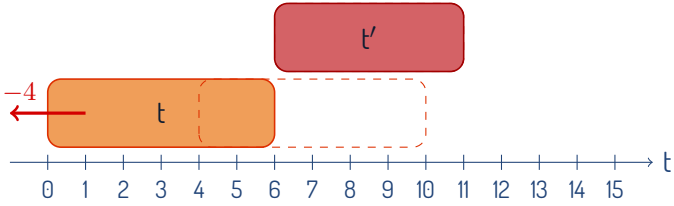


Figure – Réparation de la contrainte : avec start[t] seulement

Réparation d'une contrainte de ressource disjonctive : non déterministe

Réparation (non déterministe) d'une précédence de la disjonction :

$$(\mathbf{start}[\mathbf{t'}] \geq \mathbf{end}[\mathbf{t}]) \;||\; (\mathrm{start}[t] \geq \mathrm{end}[t'])$$

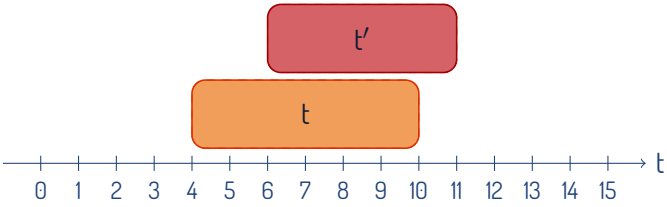


Figure – Contrainte non respectée

Réparation d'une contrainte de ressource disjonctive : non déterministe

Réparation (non déterministe) d'une précédence de la disjonction :

$$(\mathbf{start}[\mathbf{t'}] \geq \mathbf{end}[\mathbf{t}]) \;||\; (\mathrm{start}[t] \geq \mathrm{end}[t'])$$

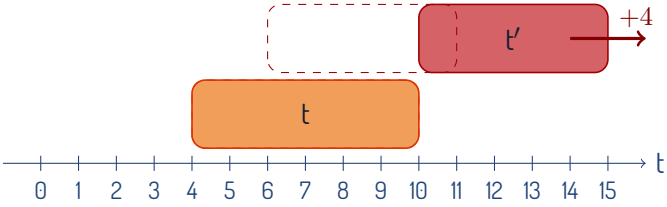


Figure – Réparation de la contrainte : avec start[t'] seulement

Réparation d'une contrainte de ressource disjonctive : non déterministe

Réparation (non déterministe) d'une précédence de la disjonction :

$$(\mathbf{start}[\mathbf{t'}] \geq \mathbf{end}[\mathbf{t}]) \;||\; (\mathrm{start}[t] \geq \mathrm{end}[t'])$$

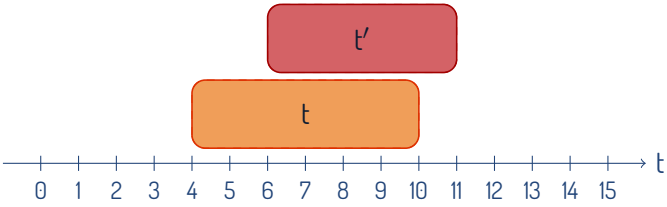


Figure – Contrainte non respectée

Réparation d'une contrainte de ressource disjonctive : non déterministe

Réparation (non déterministe) d'une précédence de la disjonction :

$$(\mathbf{start}[\mathbf{t'}] \geq \mathbf{end}[\mathbf{t}]) \;||\; (\mathrm{start}[t] \geq \mathrm{end}[t'])$$

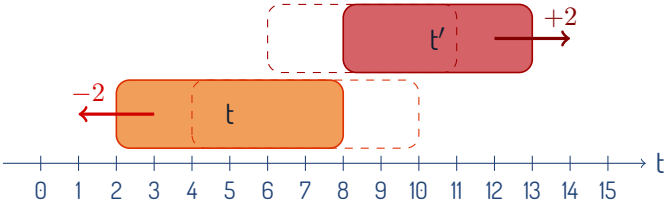


Figure – Réparation de la contrainte : équitablement entre start[t] et start[t′]

Réparation d'une contrainte de ressource disjonctive : non déterministe

Réparation (non déterministe) d'une précédence de la disjonction :

$$(\mathbf{start}[\mathbf{t'}] \geq \mathbf{end}[\mathbf{t}]) \;||\; (\mathrm{start}[t] \geq \mathrm{end}[t'])$$

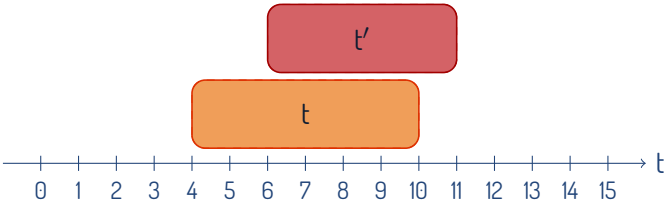


Figure – Contrainte non respectée

Réparation d'une contrainte de ressource disjonctive : non déterministe

Réparation (non déterministe) d'une précédence de la disjonction :

$$(\mathbf{start}[\mathbf{t'}] \geq \mathbf{end}[\mathbf{t}]) \;||\; (\mathrm{start}[t] \geq \mathrm{end}[t'])$$

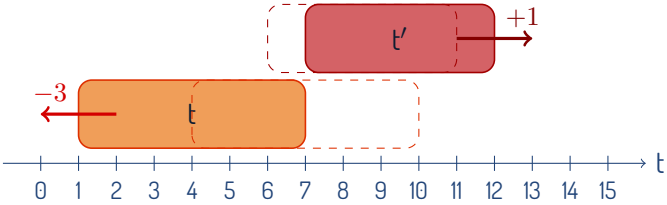


Figure – Réparation de la contrainte : aléatoirement entre start[t] et start[t′]

Autre modélisation des contraintes de ressource disjonctive : ordre sur les tâches

Formulation alternative des contraintes de ressource disjonctive, avec LocalSolver 9.5 :

```
1 order <- list(nbJobs);
2 constraint count(order) == nbJobs;
3 constraint and(0..count(order)-2, i => start[order[i+1]] >= end[order[i]]);
```

Avantages de cette formulation :

- Fonctionne aussi lorsque le nombre de tâches sur la machine n'est pas fixé
- Plus compact : $O(nm)$ contraintes contre $O(n^2m)$

Forme des contraintes propagées

Forme des contraintes propagées avec LocalSolver 9.5 :

- Contraintes de précédence (généralisées) : $aX + bY \leq c$

```
start[j][m+1] >= end[j][m];
```

- Contraintes de ressource disjonctive (généralisées)
  - tâches prises deux à deux : $\bigvee_i (a_i X_i + b_i Y_i \leq c_i)$

```
start[j1][m] >= end[j2][m] || start[j2][m] >= start[j1][m];
```

  - ordre sur les tâches : $\bigwedge_i (aX_{L[f(i)]} + bX_{L[g(i)]} \leq c_{L[h(i)]})$

```
and(1..nbJobs-1, i => start[order[i+1]][m] >= end[order[i]][m]);
```

Résultats obtenus

Résultats – Jobshop, classes d'instances FT, LA et ORB

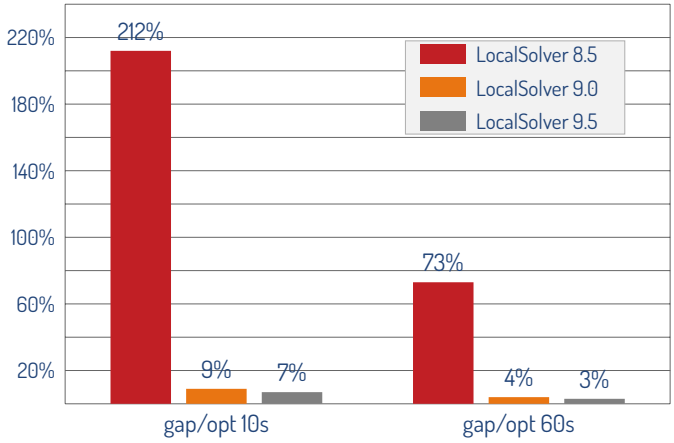


Figure – Amélioration moyenne de l'écart à l'optimum entre LocalSolver 8.5, 9.0 (modèle entiers), et 9.5 (modèle listes et entiers) sur le Jobshop

Résultats – Autres types de problèmes

LocalSolver est un solveur générique, non dédié à l'ordonnancement : les évolutions visent à être le plus génériques possibles

Ce mécanisme de réparation de solutions permet d'améliorer les performances de LocalSolver sur d'autres types de problèmes :

- Packing
- Layout
- Mining

Exemple : amélioration de ∼35% sur un problème de packing 3D de notre base de tests

Conclusion

Amélioration visée Problèmes avec réseaux de précédences généralisées

Problème avec LocalSolver 8.5 Passage d'une bonne solution à une autre :

- Beaucoup de petits changements nécessaires
- $\Rightarrow$ La recherche locale ne suffit pas

Solution mise en place Réparation par propagation des contraintes

- Réparations successives des contraintes
- Modification des variables toujours dans le même sens