LocalSolver 4.0

Hybrid Math Programming

Thierry Benoist **Julien Darlay** Bertrand Estellon
Frédéric Gardi Romain Megel

www.localsolver.com

Who are we ?

Bouygues, one of the French largest corporation, €33 bn in revenues

LocalSolver, mathematical optimization solver commercialized by Innovation 24

Agenda

LocalSolver

- Quick introduction
- Examples
- Technology
- Benchmarks
- Roadmap

LocalSolver in practice

LocalSolver

Solver for combinatorial & continuous optimization

- Provides good-quality solutions in short running times
- Allows to tackle large-scale problems
- Simple mathematical modeling formalism
  - C++, Java, .NET APIs
  - Modeling Language (LSP)

Solver based on local search

- Moves based on decisions/constraints hypergraph
- Incremental evaluation: millions of moves per minute
- Adaptive, randomized, parallelized simulated annealing with restarts

Free academic licenses

Commercial licenses from 990 €

LocalSolver

LocalSolver 4.0

Mathematical programming solver

- **For combinatorial optimization**
- For numerical optimization
- For mixed-variable optimization

- **Provides solutions (upper bounds)**
- Provides lower bounds
- Infeasibility gap/proof, optimality gap/proof

Suited for large-scale non-convex optimization

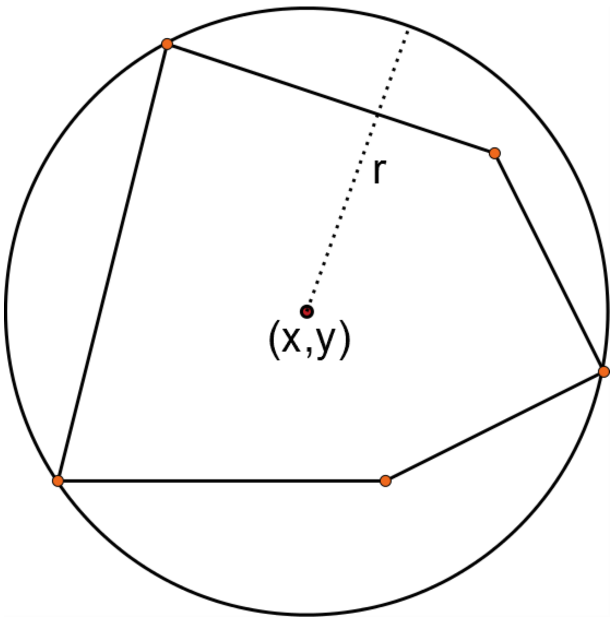
- Millions of combinatorial and/or continuous variables
- Non-convex constraints and/or objectives
- Short resolution times

Examples

Numerical optimization

Smallest Circle

- Find the circle of minimum radius including a set of points
- Two continuous decisions: ***x*** and ***y***
- The radius ***r***: expression deduced from decisions
- Straightforward quadratic model



Continuous decision

Quadratic expression

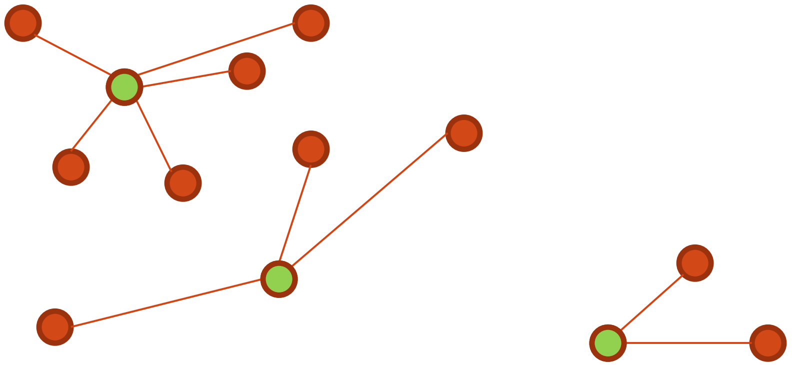
```
x <- float(minX, maxX);
y <- float(minY, maxY);

r2 <- max[i in 1..n] (pow(x-coordX[i],2) + pow(y-coordY[i],2));

minimize sqrt(r2);
```

LocalSolver

Constrained combinatorial optimization

P-Median

- Find a subset of P elements in a set of N
- Minimize the sum of distances from each element to the closest one in P



```
function model() {
    x[1..N] <- bool();
    constraint sum[i in 1..N] (x[i]) == P;

    minDistance[i in 1..N] <- min[j in 1..N]  (x[j] ? distance[i][j] : +inf);
    minimize sum[i in 1..N] (minDistance[i]);
}
```

LocalSolver

Local Search

Local Search

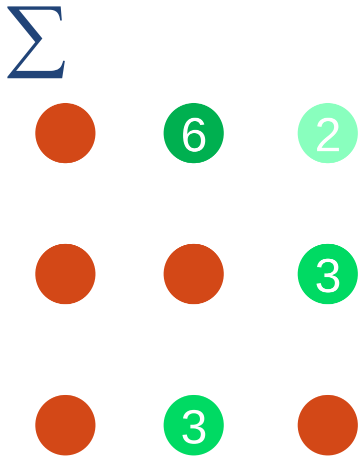
Main idea for combinatorial optimization

- Sequential modification of a small number of decisions
- Maintaining the feasibility of current solution
- Incremental evaluation, generally in $O(1)$ time

→ Small improvement probability but small time and space complexity

A three layers architecture

- Moves based on mathematical model
- Incremental evaluation of solutions
- Heuristic to drive the search



Moves

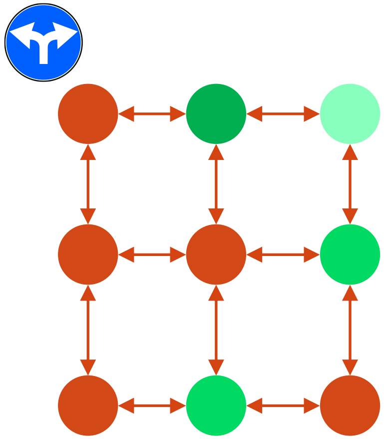
Standard moves in combinatorial optimization: “k-flips”

- Could lead to infeasible solution on real instances
- If feasibility is hard to reach: slow convergence

LocalSolver maintains feasibility

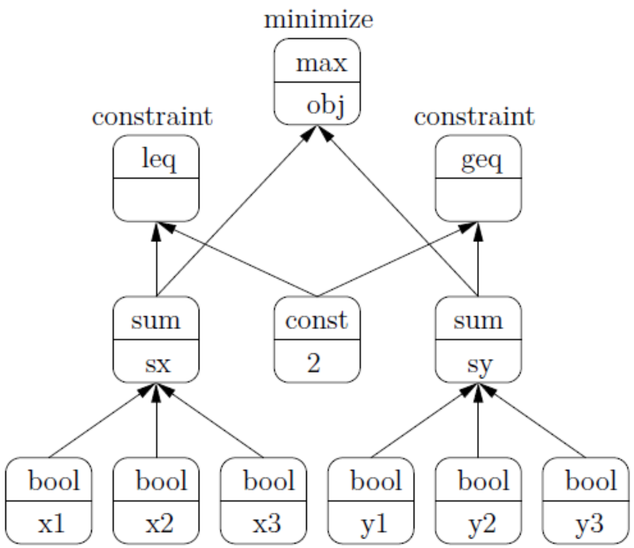
- « Destroy & repair »
- Ejection chain on constraint graph
- Use of known combinatorial structure

```
...
x[1..N] <- bool();
constraint sum[i in 1..N] (x[i]) == P;
...
```



Fast exploration

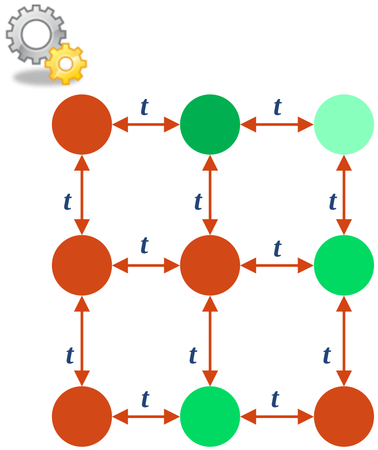
```
x1 <- bool();
x2 <- bool();
x3 <- bool();
y1 <- bool();
y2 <- bool();
y3 <- bool();
sx <- sum(x1, x2, x3);
sy <- sum(y1, y2, y3);
constraint leq(sx, 2);
constraint geq(sy, 2);
obj <- max(sx, sy);
minimize obj;
```



Incremental evaluation

- “Lazy” propagation in the expression DAG
- Usage of invariants

⟶ Millions of moves per minute



Heuristic

Online learning of moves

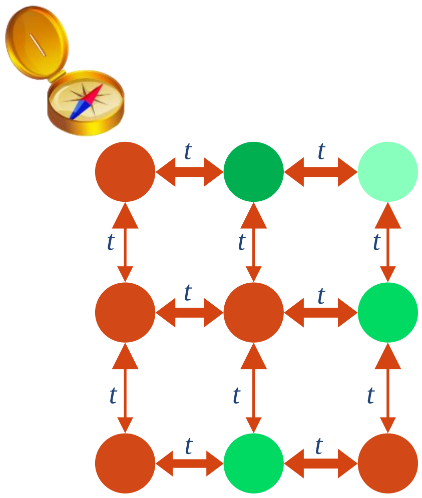
- Discard inefficient moves
- Improve efficient moves selection

Simulated annealing

- Handle non smooth objectives
- Allow degrading solutions

« Restart » + parallel search

- Avoid local optima
- Improve search space coverage



LocalSolver

Benchmarks

Combinatorial optimization

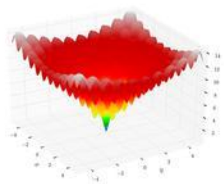
Car Sequencing : schedule cars among an assembly line

10 sec	100	200	300	400	500
Gurobi 5.5	140	274	X	429	513
LocalSolver 4.0	8	5	8	10	19
60 sec	**100**	**200**	**300**	**400**	**500**
Gurobi 5.5	3	66	1	356	513
LocalSolver 4.0	6	4	3	5	6
600 sec	**100**	**200**	**300**	**400**	**500**
Gurobi 5.5	3	2	*0	1	20
LocalSolver 4.0	4	*0	*0	2	*0

Non constrained non convex optimization

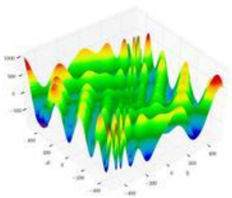
Quasi optimal solutions in a few seconds on several artificial landscapes from the literature

Oldenhuis (2009). Test functions for global optimization algorithms. Matlab



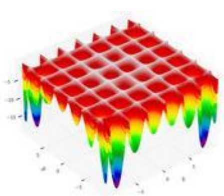
$$f(x,y) = -20\exp\left(-0.2\sqrt{0.5\left(x^2+y^2\right)}\right) - \exp\left(0.5\left(\cos\left(2\pi x\right)+\cos\left(2\pi y\right)\right)\right) + 20 + e.$$

gap (%) < 10^{-6}



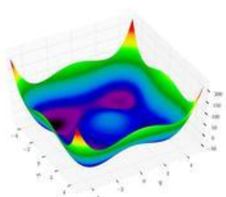
$$f(x,y) = -\left(y+47\right)\sin\left(\sqrt{\left|y+\frac{x}{2}+47\right|}\right) - x\sin\left(\sqrt{\left|x-\left(y+47\right)\right|}\right).$$

gap (%) < 10^{-4}



$$f(x,y) = -\left|\sin\left(x\right)\cos\left(y\right)\exp\left(\left|1-\frac{\sqrt{x^2+y^2}}{\pi}\right|\right)\right|.$$

gap (%) < 10^{-4}



$$f(\boldsymbol{x}) = \frac{\sum_{i=1}^{n} x_i^4 - 16x_i^2 + 5x_i}{2}.$$

n = 10 → 10000

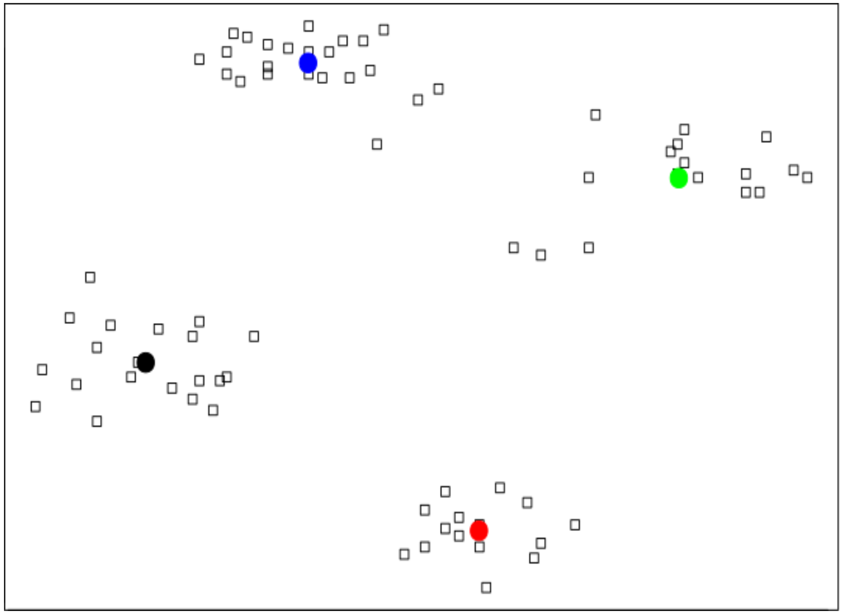
gap (%) < 10^{-6} → 10^{-1}

LocalSolver

Combinatorial / Continuous optimization

K-means

- Machine learning problem
- NP-Hard, Quadratic
- Same solutions in comb. or cont.



Instance	k	OPT*	LS 4.0	GAP
ruspini	2	89337	89337,9	0,00%
	3	51063,4	51063,5	0,00%
	4	12881	12881,1	0,00%
	5	10126,7	10126,8	0,00%
	6	8575,41	8670,86	1,11%
	7	7126,2	7159,13	0,46%
	8	6149,64	6158,26	0,14%
	9	5181,64	5277,11	1,84%
	10	4446,28	4856,98	9,24%
iris	2	152,348	152,369	0,01%
	3	78,8514	78,9412	0,11%
	4	57,2285	57,3556	0,22%
	5	46,4462	46,5363	0,19%
	6	39,04	41,7964	7,06%
	7	34,2982	34,6489	1,02%
	8	29,9889	30,3029	1,05%
	9	27,7861	28,0667	1,01%
	10	25,834	26,0521	0,84%
glass	20	114,646	120,048	4,71%
	30	63,2478	74,1251	17,20%
	40	39,4983	58,3912	47,83%
	50	26,7675	52,4679	96,01%

*[Aloise et al. 2012]

Toward an “all-in-one” solver

One solver to tackle all kinds of problems

- Discrete, numerical, or mixed-variable optimization
- From small-scale to large-scale problems
- Best effort to prove infeasibility or optimality
- Able to scale heuristically faced with large problems

One solver offering the best of all optimization techniques

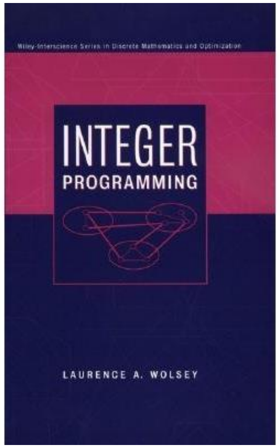
- Local and direct search
- Constraint propagation and inference
- Linear and mixed-integer programming
- Nonlinear programming (convex and non-convex)
- Dynamic programming
- Specific algorithms: paths, trees, flows, matchings, etc.

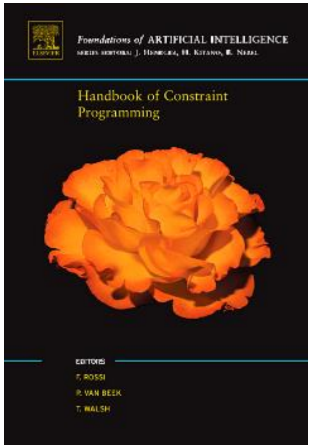
LocalSolver in practice

How to migrate to LocalSolver

Modeling patterns ?

A classic topic in MIP or CP





Very little literature on modeling for Local Search…

…**because of** the absence of model-and-run solver

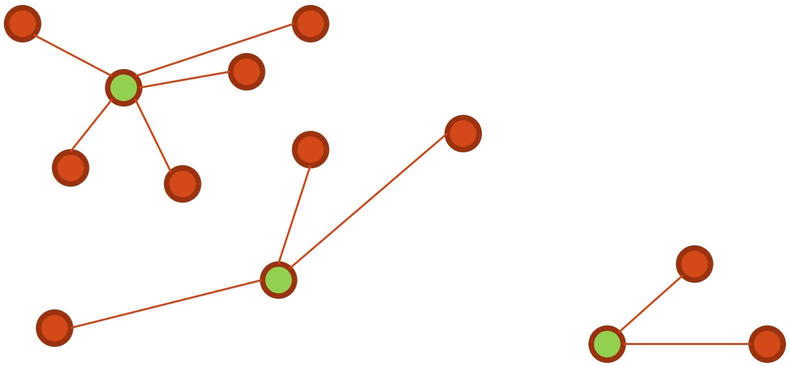
→ models and algorithms were designed together and not always clearly separated

Modeling Pattern #1

Choose the right set of decision variables

Choose the right set of decision variables

Select a set S of P cities among N
Minimizing the sum of distances
from each city to the closest city in S

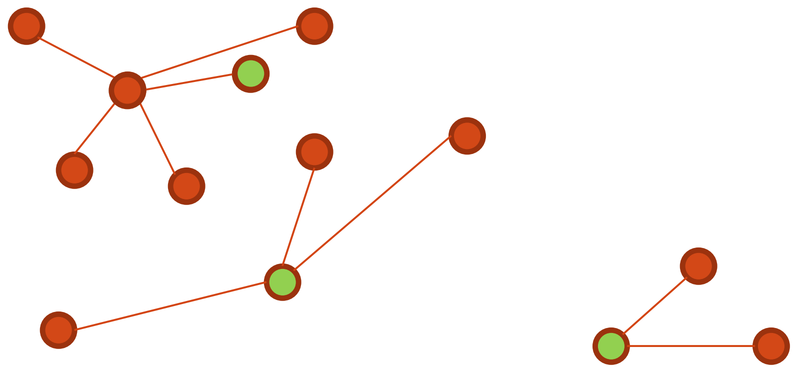


```
function model() {
   X[1..N] <- bool();  // x[i] = 1 if hospital in city i
   Y[1..N][1..N] <- bool(); // y[i][j] = 1 if city i is assigned to hospital j

   for[i in 1..N]
            constraint sum[j in 1..N] Y[i][j] == 1;

   for[i in 1..N][j in 1..N]
            constraint Y[i][j] <= X[j];

   constraint sum[i in 1..N] (X[i]) == P;

   minimize sum[i in 1..N][j in 1..N] (distance[i][j]*Y[i][j]);
}
```

Given the values of variables Y, I can infer the values of X
Given the values of variables X, I can infer the values of Y

Too many decision variables!

First attempt: keep only decision variables Y



Select a set S of P cities among N
Minimizing the sum of distances
from each city to the closest city in S

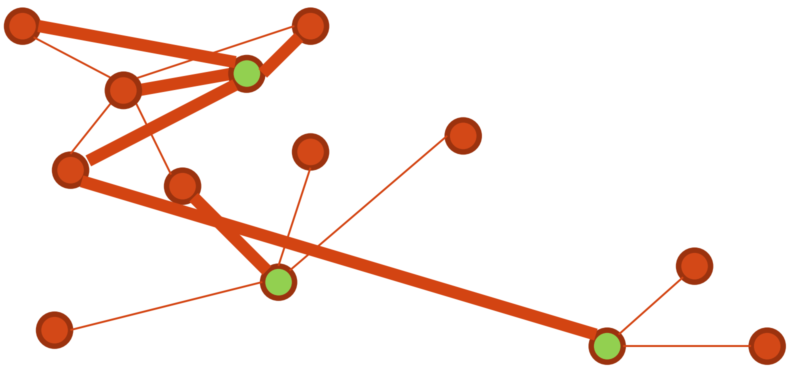
```
function model() {
    X[1..N] <- bool(); // x[i] = 1 if hospital in city i
    Y[1..N][1..N] <- bool(); // y[i][j] = 1 if city i is assigned to hospital j

    for[i in 1..N]
            constraint sum[j in 1..N] Y[i][j] == 1;

    for[j in 1..N]
            X[j] <- or[j in 1..N] (Y[i][j]);

    constraint sum[i in 1..N] (X[i]) == P;

    minimize sum[i in 1..N][j in 1..N] (distance[i][j]*Y[i][j]);
}
```

We can introduce non-decision expressions

We can use non-linear operators

How many variables need to be changed to move from a feasible solution to another feasible solution ?

First attempt: keep only decision variables Y



Select a set S of P cities among N
Minimizing the sum of distances
from each city to the closest city in S

```
function model() {
    X[1..N] <- bool(); // x[i] = 1 if hospital in city i
    Y[1..N][1..N] <- bool(); // y[i][j] = 1 if city i is assigned to hospital j

    for[i in 1..N]
            constraint sum[j in 1..N] Y[i][j] == 1;

    for[j in 1..N]
            X[j] <- or[j in 1..N] (Y[i][j] <= X[j];

    constraint sum[i in 1..N] (X[i]) == P;

    minimize sum[i in 1..N][j in 1..N] (distance[i][j]*Y[i][j]);
}
```

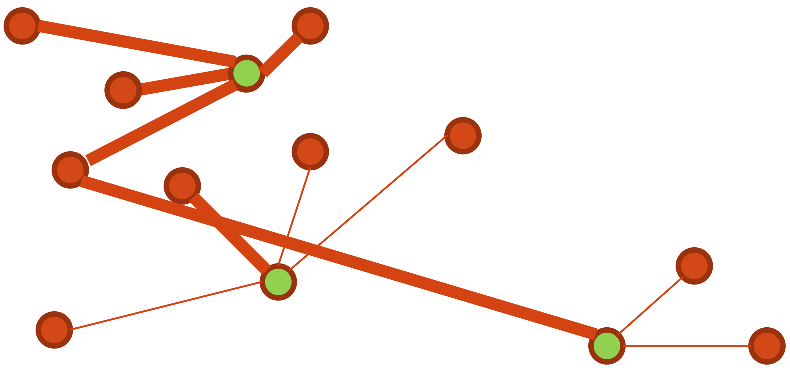
We can introduce non-decision expressions

We can use non-linear operators

How many variables need to be changed to move from a feasible solution to another feasible solution ?

First attempt: keep only decision variables Y

Select a set S of P cities among N
Minimizing the sum of distances
from each city to the closest city in S



```
function model() {
    X[1..N] <- bool(); // x[i] = 1 if hospital in city i
    Y[1..N][1..N] <- bool(); // y[i][j] = 1 if city i is assigned to hospital j

    for[i in 1..N]
            constraint sum[j in 1..N] Y[i][j] == 1;

    for[j in 1..N]
            X[j] <- or[j in 1..N] (Y[i][j] <= X[j];

    constraint sum[i in 1..N] (X[i]) == P;

    minimize sum[i in 1..N][j in 1..N] (distance[i][j]*Y[i][j]);
}
```

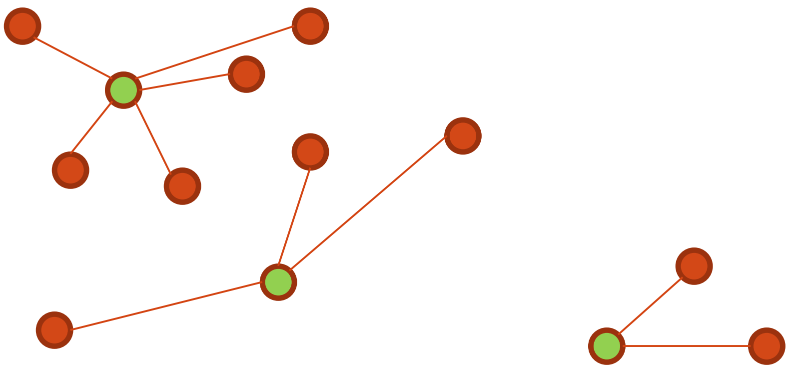
We can introduce non-decision expressions

We can use non-linear operators

How many variables need to be changed to move from a feasible solution to another feasible solution ?

Second attempt: keep only decision variables X

Select a set S of P cities among N
Minimizing the sum of distances
from each city to the closest city in S



```
function model() {
    x[1..N] <- bool();
    constraint sum[i in 1..N] (x[i]) == P;

    minDistance[i in 1..N] <- min[j in 1..N]  (x[j] ? distance[i][j] : +inf);
    minimize sum[i in 1..N] (minDistance[i]);
}
```

Even conditional expressions are allowed !

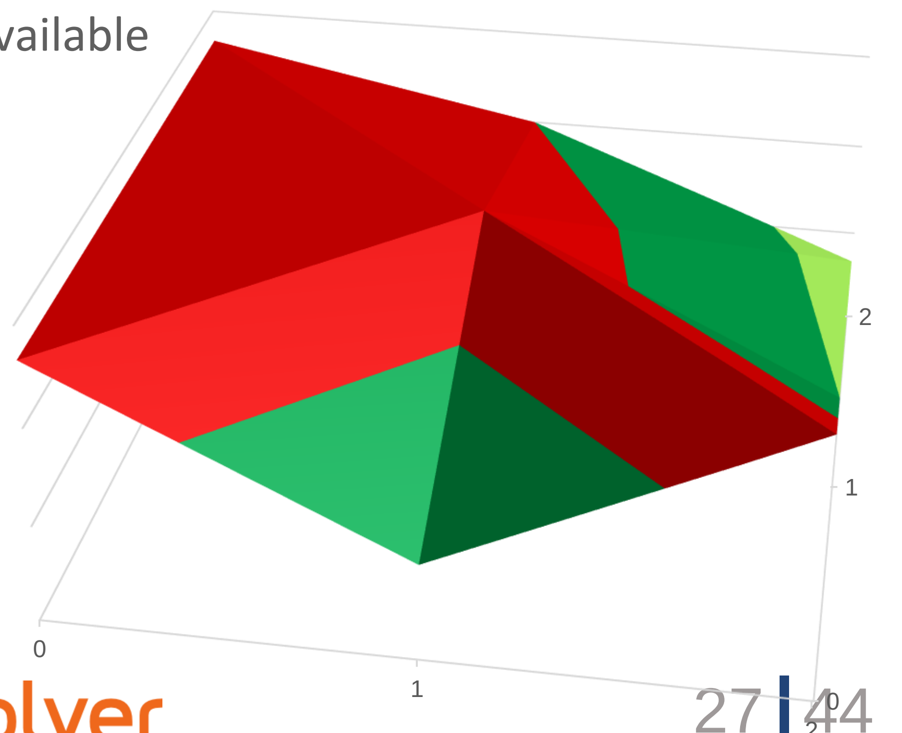
- Now the hamming distance between two feasible solutions is 2
- We have only N decision variables

Modeling Pattern #1

Choose the right set of decision variables

A good model defines a good search space

- Intermediate variables are inferred from decisions
- Logical and non-linear expressions available
- No artificial variables needed



LocalSolver

Modeling Pattern #2

Do not limit yourself to linear operators

LocalSolver

Do not limit yourself to linear operators

TRAVELING SALESMAN PROBLEM

MIP approach: X_{ij}=1 if city j is after city i in the tour

- Matching constraints $\sum_j X_{ij} = 1$ and $\sum_i X_{ij} = 1$
- Plus an exponential number of subtour elimination constraints
- Minimize $\sum_{ij} c_{ij} X_{ij}$

Polynomial non-linear model: : X_{ik}=1 if city *i* is in position *k* in the tour

- Matching constraints $\sum_k X_{ik} = 1$ and $\sum_i X_{ik} = 1$
- $Y_k \leftarrow \sum_i iX_{ik}$ the index of the kth city of the tour
- Minimize $\sum_k c_{[Y_k, Y_{k+1}]}$

"at" operator

TSP Lib: average gap after 10mn = 2.6%

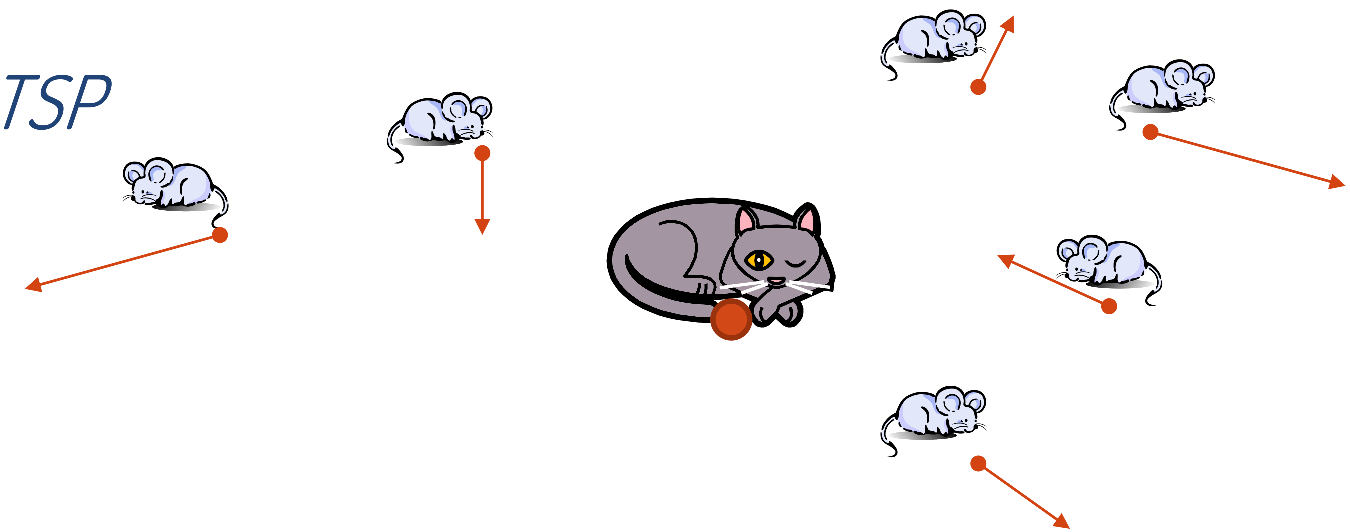
Why solving a TSP with LocalSolver ?

$$\mathrm{Time}[A, B, T] =$$

$$\frac{2\left({\alpha_x}^2+{\alpha_y}^2\right)}{-2\left(\alpha_x\beta_x+\alpha_y\beta_y\right)+\sqrt{4\left(\alpha_x\beta_x+\alpha_y\beta_y\right)^2-4\left({\alpha_x}^2+{\alpha_y}^2\right)\left({\beta_x}^2+{\beta_y}^2-V^2\right)}} + T$$

With $\alpha_x, \beta_x, \alpha_y, \beta_y$ function of A,B and T

Kinetic TSP



Modeling Pattern #2

Do not limit yourself to linear operators

Even arrays are allowed (indexing operator)

- Yield very compact models
- Can model any relation between two variables

Modeling Pattern #3

Precompute what can be precomputed

LocalSolver

Precompute what can be precomputed

Document processing : in a table, a text cell has several possible *height x width* configurations .

LocalSolver : mathematical programming by local search

29 x 82

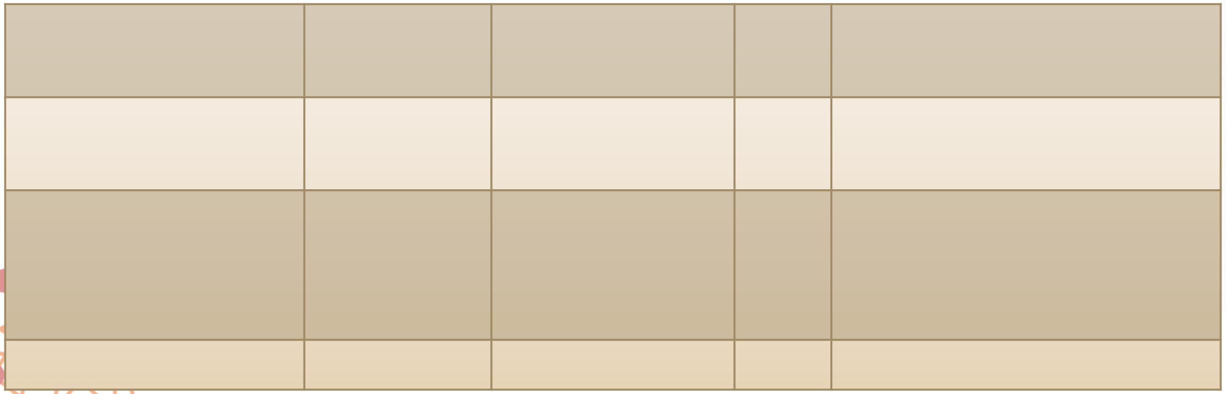
LocalSolver : mathematical programming by local search

34 x 61

LocalSolver : mathematical programming by local search

45 x 43

Select a configuration for each cell, in order to minimize the height of the table (whose width is limited)

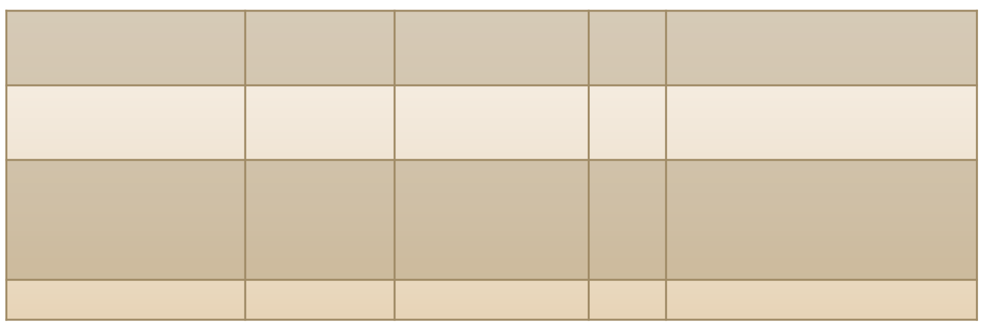


Precompute what can be precomputed

First model : 1 decision variable per possible configuration for each cell

Extended formulation :

- Note that from the width of a column you can infer the minimum height of each of its cells.
- 1 decision variable per possible width per column
- Consequence: by changing a single decision variable, LocalSolver will update the height and width of all cells in the column



Modeling Pattern #3

Precompute what can be precomputed

Similar to column generation models

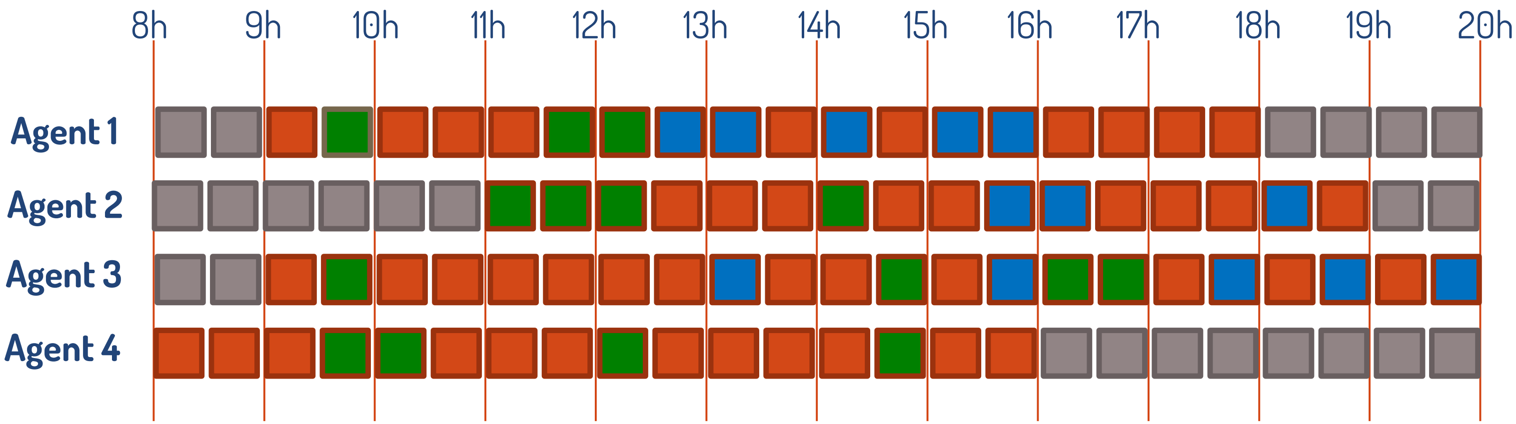
- Yields very dense search space

Modeling Pattern #4

Separate commands and effects

Separate commands and effects

Multi-skill workforce scheduling



Candidate model

$\text{Skill}_{atk} = 1 \Leftrightarrow$ agent a works on skill k at timestep t

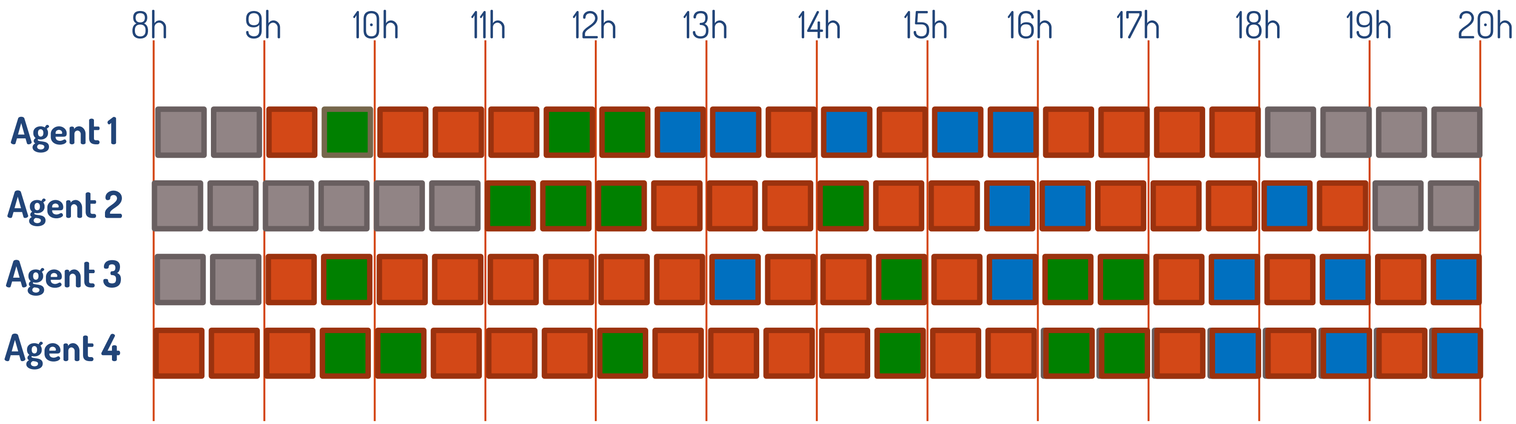
Constraint $SUM_k\,(\text{Skill}_{atk}) <= 1$

Constraint $OR_k\,(\text{Skill}_{atk}) == (t \in [\text{Start}_a, \text{End}_a[\ )$

Problem: any change of $Start_a$ will be rejected unless skills are updated for all impacted timesteps

Separate commands and effects

Multi-skill workforce scheduling



Alternative model

$\text{SkillReady}_{atk} = 1 \Leftrightarrow$ agent a will works on skill k at timestep t **if present**

Constraint SUM_k $(\text{SkillReady}_{atk}) == 1$

$\text{Skill}_{atk} \leftarrow \text{AND}(\text{SkillReady}_{atk}\ ,\ t \in [\text{Start}_a, \text{End}_a[)$

Now we have no constraint between skills and worked hours

-> for any change of $Start_a$ skills are automatically updated

Modeling Pattern #4

Separate commands and effects

Can avoid defining strong constraints between related variables

-> minimize the hamming distance between feasible solutions

Similar case: Unit Commitment Problems

- A generator is active or not, but when active the production is in $[P_{min}, P_{max}]$
- Better modeled without any constraint

$$\begin{aligned}&\text{ProdReady}_{gt} \leftarrow \text{float}(P_{min}, P_{max})\\&\text{Active}_{gt} \leftarrow \text{bool}()\\&\text{Prod}_{gt} \leftarrow \text{Active}_{gt} \times \text{ProdReady}_{gt}\end{aligned}$$

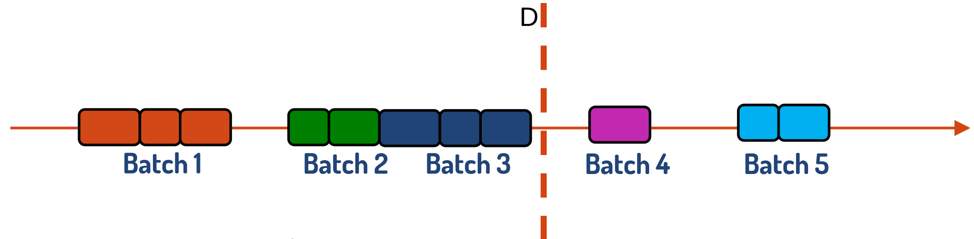
Modeling Pattern #5

Use dominance properties

Use dominance properties

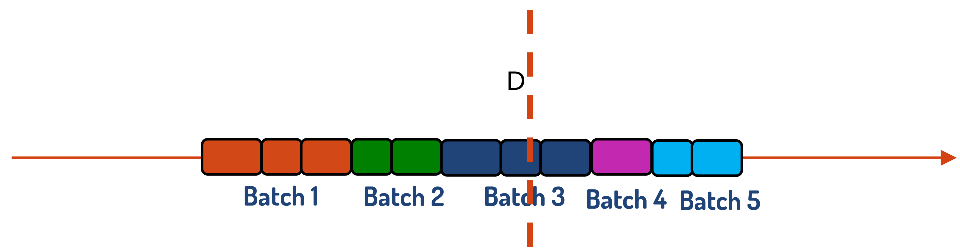
Batch scheduling for *N* jobs having the same due date D.

- Completion time of each job will be that of the batch selected for this job
- Linear late or early cost (α_k β_k)



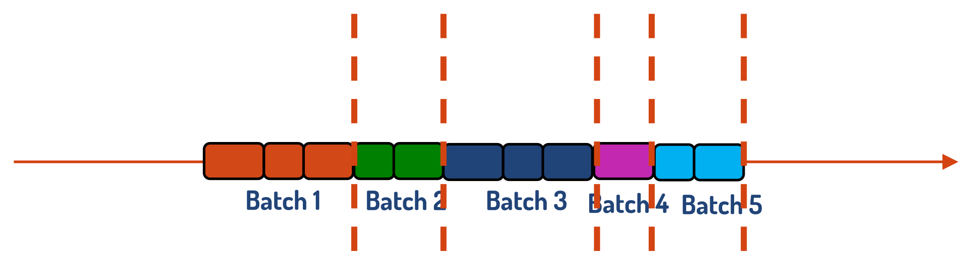
Basic Model

Date variable for each batch
+ assignment of jobs to batches
+ Precedence constraints



No idle time

Only one starting date variable
+ assignment of jobs to batches



Optimal start will position due date at the end of a batch

No start date variable
+ assignment of jobs to batches
+ penalty[k] if due date at the end of batch k
Minimize min[k in 1..5](penalty[k])

We can **minimize a minimum**

As if starting date was automatically adjusted after each move

Summary

1. Choose the right set of decision variables
2. Do not limit yourself to linear operators
3. Precompute what can be precomputed
4. Separate commands and effects
5. Use dominance properties

Modeling Pattern #6

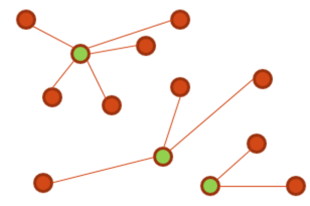
Your turn!

Conclusion

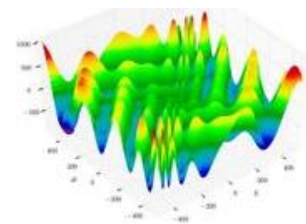
Hybrid math programming solver

LocalSolver 4.0
=
LS + CP/SAT + LP/MIP + NLP

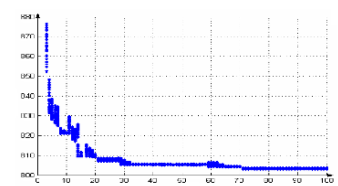
For combinatorial, numerical,
or mixed-variable optimization



Particularly suited for large-scale
non-convex optimization



High-quality solutions in seconds
without tuning



Free for academics, Business licenses from 990 €,

COME AND MEET US ON EXHIBITION BOOTH 21

LocalSolver

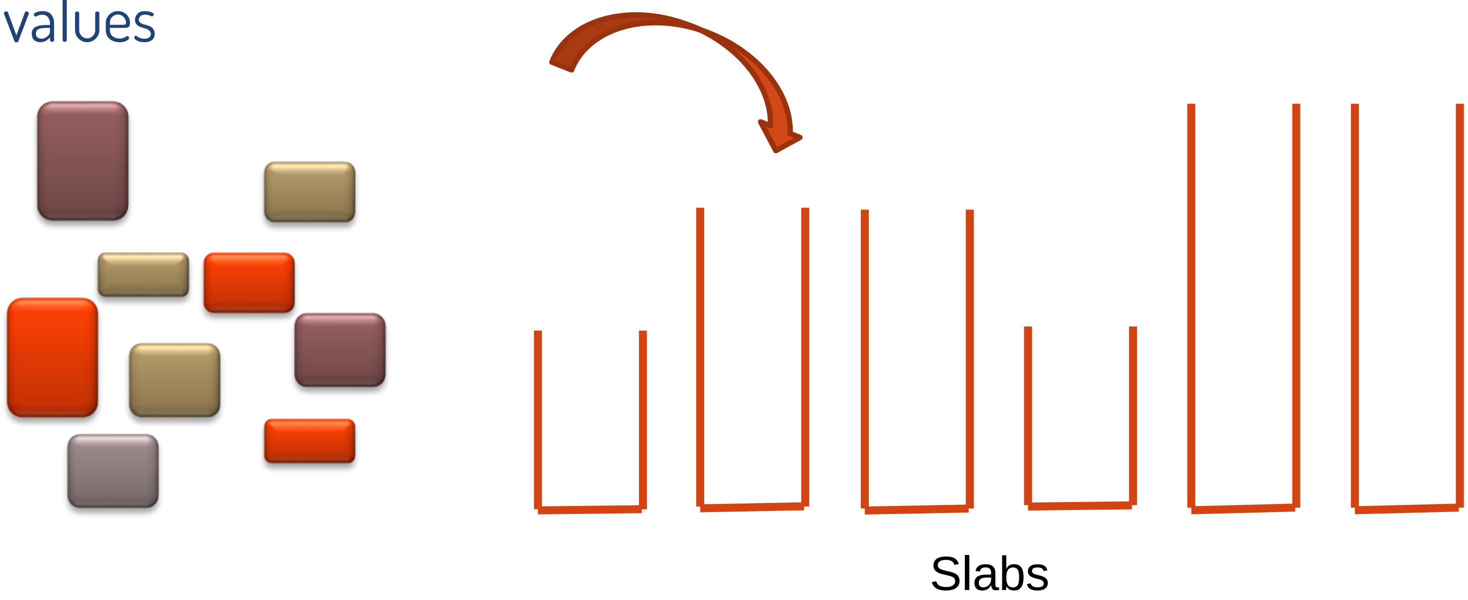
Thierry Benoist **Julien Darlay** Bertrand Estellon
Frédéric Gardi Romain Megel

www.localsolver.com

Choose the right set of decision variables

Industrial « Bin-packing »

Assignment of steel orders to « slabs » whose capacity can take only 5 different values



Model

$X_{ij} = 1 \Leftrightarrow$ Order i is assigned to slab j

$Y_{jk} = 1 \Leftrightarrow$ Slab j takes capacity k

Minimize total size of slabs

Choose the right set of decision variables

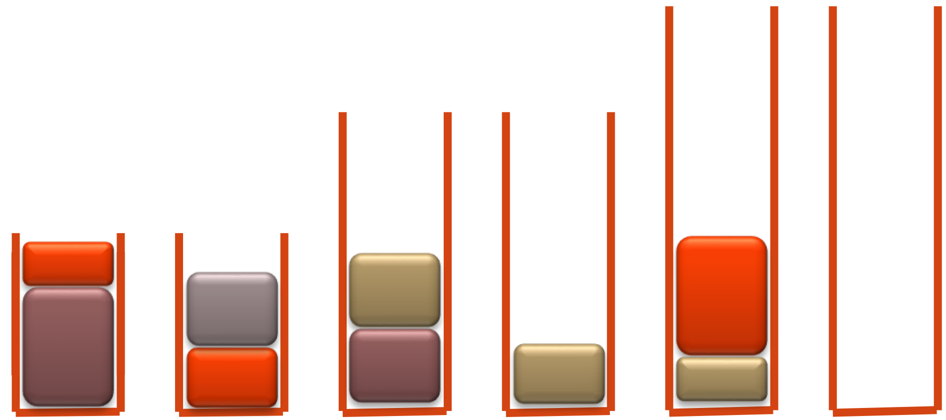
Industrial « Bin-packing »

Assignment of steel orders to « slabs » whose capacity can take only 5 different values

Change

Exchange

Ejection chain



Slabs

In a good model:

- when a value can be computed from others it is defined with operator <- (it is an **intermediate** variable)
- moving from a feasible solution to another feasible solution only requires modifying a small number of **decision** variables.

Choose the right set of decision variables

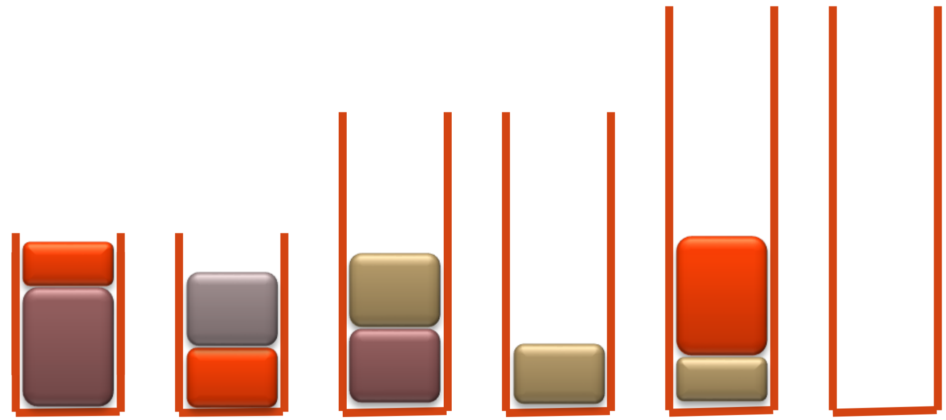
Industrial « Bin-packing »

Assignment of steel orders to « slabs » whose capacity can take only 5 different values

Change

Exchange

Ejection chain



Slabs

Model

$X_{ij} = 1 \Leftrightarrow$ Order i is assigned to slab j

$Capa_k \leftarrow MinCapa[content_k]$

"at" operator

LocalSolver