Solving the Assembly Line Balancing Problem with LocalSolver

Léa Blaise

lblaise@localsolver.com

www.localsolver.com

OR 2023

Presentation of LocalSolver

"**Model and run**" optimization solver

- Simple non-linear and set-based formalism
- High quality solutions in short running times, even on large instances
- Combinatorial, continuous and mixed problems

Global solver: efficient and reliable optimization techniques

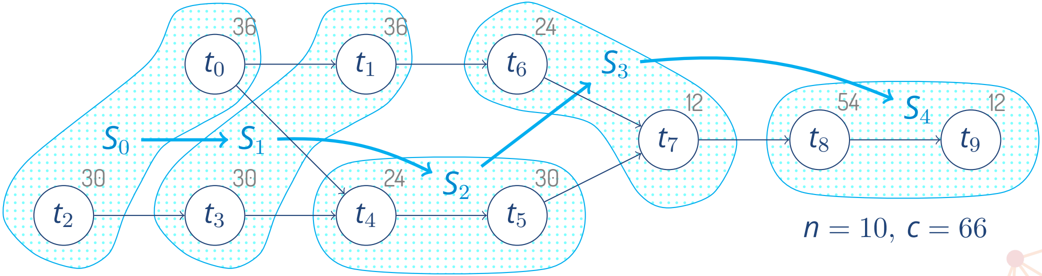
- Simplex algorithm, interior points algorithm, branch and bound, propagation...
- **Local search**, constructive algorithms, ...

LocalSolver model

for the Assembly Line Balancing Problem

Description of the Assembly Line Balancing Problem (SALB-1)

- n tasks to assign, n possible workstations
- Precedence relations between the tasks
- Station time must not exceed cycle time c
- Objective = minimize the number of used workstations



$n = 10,\ c = 66$

Credit: Armin Scholl (https://assembly-line-balancing.de)

Set variables

Set variable of domain size n = subset of { 0, 1, ..., n-1 }

```
mySetVariable <- set(n);
```

Characteristics:

- Value $\neq$ single number
- Value = set of numbers
- Each element is unique
- Variable size
- Unordered

Operators:

- count
- contains
- partition
- find
- lambda-functions

Examples for n=5:

- { }
- { 1 }
- { 0, 1, 4 }
- { 0, 1, 2, 3, 4 }

LSP model for the Assembly Line Balancing Problem

```
function model() {
    stations[s in 0..maxNbStations-1] <- set(nbTasks);
    constraint partition(stations);
    chosenStation[t in 0..nbTasks-1] <- find(stations, t);
    for [t in 0..nbTasks-1][succ in successors[t]] {
        constraint chosenStation[t] <= chosenStation[succ];
    }
    for [s in 0..maxNbStations-1] {
        stationTime[s] <- sum(stations[s], t => duration[t]);
        constraint stationTime[s] <= cycleTime;
    }
    stationUsed[s in 0..maxNbStations-1] <- count(stations[s]) > 0;
    nbStations <- sum[s in 0..maxNbStations-1] (stationUsed[s]);
    minimize nbStations;
}
```

LSP model for the Assembly Line Balancing Problem

```
function model() {
    stations[s in 0..maxNbStations-1] <- set(nbTasks);
    constraint partition(stations);
    chosenStation[t in 0..nbTasks-1]
    for [t in 0..nbTasks-1][succ in s
        constraint chosenStation[t] <= chosenStation[succ];
    }
    for [s in 0..maxNbStations-1] {
        stationTime[s] <- sum(stations[s], t => duration[t]);
        constraint stationTime[s] <= cycleTime;
    }
    stationUsed[s in 0..maxNbStations-1] <- count(stations[s]) > 0;
    nbStations <- sum[s in 0..maxNbStations-1] (stationUsed[s]);
    minimize nbStations;
}
```

Set variables: set of tasks assigned to each station

LSP model for the Assembly Line Balancing Problem

```
function model() {
    stations[s in 0..maxNbStations-1] <- set(nbTasks);
    constraint partition(stations);
    chosenStation[t in 0..nbTasks-1] <- find(stations, t);
    for [t in 0..nbTasks-1][succ in su
        constraint chosenStation[t] <=
    }
    for [s in 0..maxNbStations-1] {
        stationTime[s] <- sum(stations[s], t => duration[t]);
        constraint stationTime[s] <= cycleTime;
    }
    stationUsed[s in 0..maxNbStations-1] <- count(stations[s]) > 0;
    nbStations <- sum[s in 0..maxNbStations-1] (stationUsed[s]);
    minimize nbStations;
}
```

Each task is assigned to exactly one workstation

LSP model for the Assembly Line Balancing Problem

`chosenStation[t]` is the index of the workstation executing task *t*

```
function model() {
    stations[s in 0
    constraint partition(stations);
    chosenStation[t in 0..nbTasks-1] <- find(stations, t);
    for [t in 0..nbTasks-1][succ in successors[t]] {
        constraint chosenStation[t] <= chosenStation[succ];
    }
    for [s in 0..maxNbStations-1] {
        stationTime[s] <- sum(statio
        constraint stationTime[s] <=
    }
    stationUsed[s in 0..maxNbStations-1] <- count(stations[s]) > 0;
    nbStations <- sum[s in 0..maxNbStations-1] (stationUsed[s]);
    minimize nbStations;
}
```

Each task must be scheduled before its successors

LSP model for the Assembly Line Balancing Problem

```
function model() {
    stations[s in 0..maxNbStations-1] <- set(nbTasks);
    constraint partition(stations);
    chosenStation[
    for [t in 0..n
        constraint
    }
    for [s in 0..maxNbStations-1] {
        stationTime[s] <- sum(stations[s], t => duration[t]);
        constraint stationTime[s] <= cycleTime;
    }
    stationUsed[s in 0..maxNbStations-1] <- count(stations[s]) > 0;
    nbStations <- sum[s in 0..maxNbStations-1] (stationUsed[s]);
    minimize nbStations;
}
```

Cycle time constraints (using a lambda-function) $\Longleftrightarrow \forall S, \sum_{t \in S} d_t \leq c$

LSP model for the Assembly Line Balancing Problem

```
function model() {
    stations[s in 0..maxNbStations-1] <- set(nbTasks);
    constraint partition(stations);
    chosenStation[t in 0..nbTasks-1] <- find(stations, t);
    for [t in 0..nbTasks-1][succ in successors[t]] {
        constraint chosenStation[t] <= chosenStation[succ];
    }
    for [s in 0..maxNbStations-1] {
        stationTime[s] <- sum(stations[s],
        constraint stationTime[s] <= cycleT
    }
    stationUsed[s in 0..maxNbStations-1] <- count(stations[s]) > 0;
    nbStations <- sum[s in 0..maxNbStations-1] (stationUsed[s]);
    minimize nbStations;
}
```

Minimize the number of used workstations

Packing move

based on ejection chains

Principle of the local move

Local move respecting the capacity constraints on the set variables (cycle time constraints on the workstations)

- Applicable to any problem with a packing structure

```
    stationTime[s] <- sum(stations[s], t => duration[t]);
    constraint stationTime[s] <= cycleTime;
```

LocalSolver

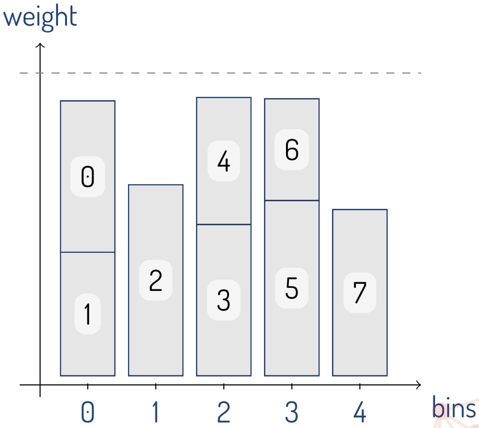
Principle of the local move

Based on ejection chains

- Series of elementary transformations: move elements from one set variable to another

Goal of the move

- Reorganize the elements present inside *k* set variables so as to empty one of them
- Help LocalSolver get out of local minima



Description of the local move

- Select a subset of non empty set variables
- Let S be the selected set variable with the lowest weight
- A random element t is ejected from S
- If there exists $S' \neq S$ in which t can be inserted : success
- Otherwise, let t' be the smallest element smaller than t that can be replaced by t
    - If t' exists, it is ejected from its set variable, t is inserted in its place, and we can start over
    - Otherwise, the move fails

Application of the local move on a small example

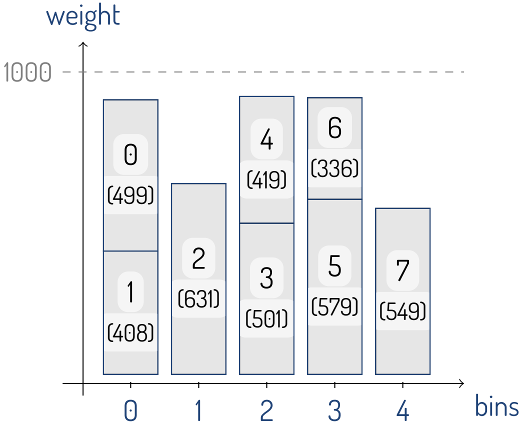


Figure 1: Initial solution

Application of the local move on a small example

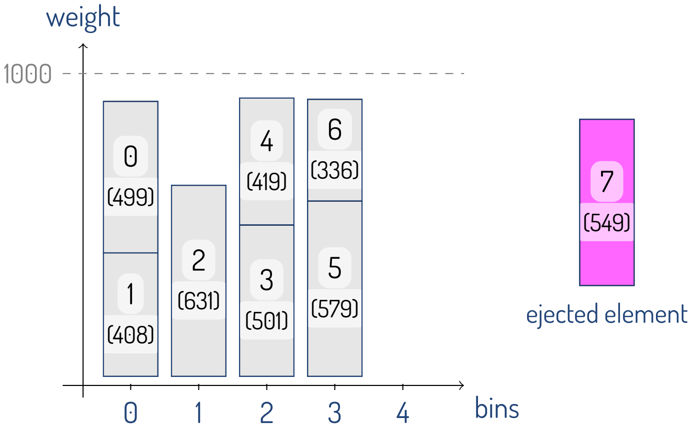


Figure 1: Element 7 is ejected from bin 4

LocalSolver

Application of the local move on a small example

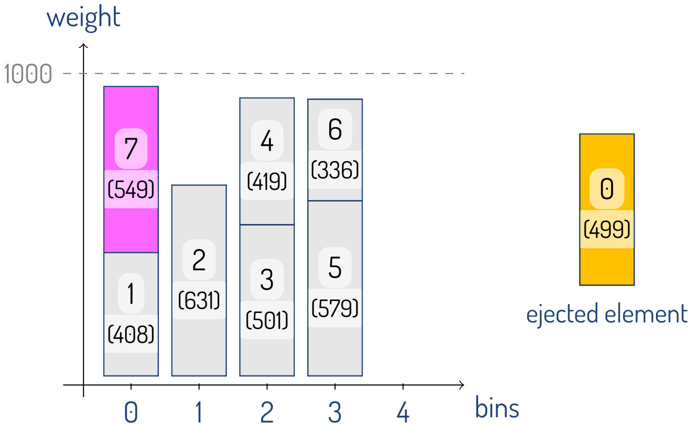


Figure 1: Element 0 is ejected from bin 0 to insert element 7

Application of the local move on a small example

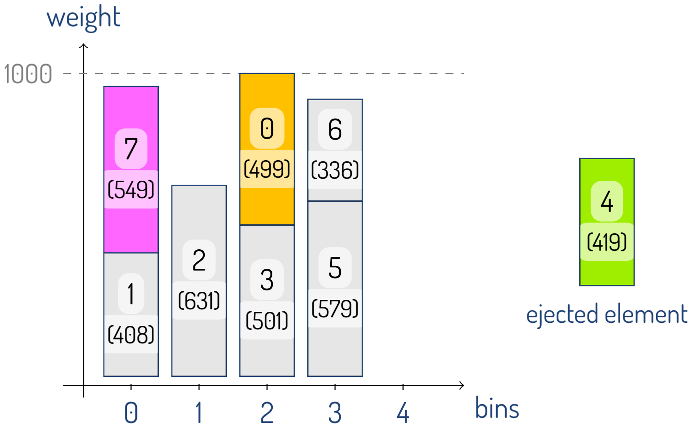


Figure 1: Element 4 is ejected from bin 2 to insert element 0

LocalSolver

Application of the local move on a small example

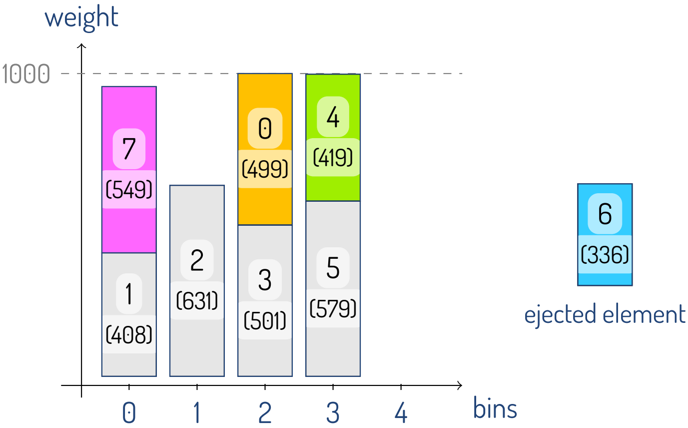


Figure 1: Element 6 is ejected from bin 3 to insert element 4

LocalSolver

Application of the local move on a small example

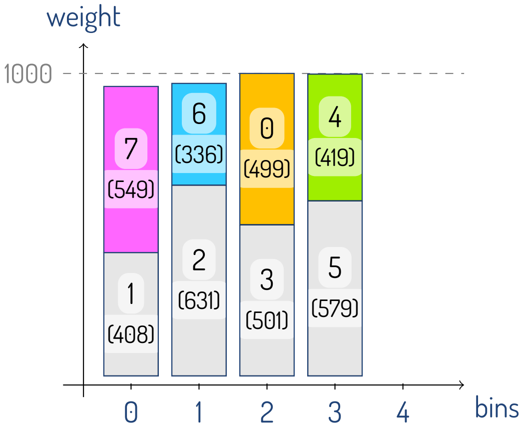


Figure 1: Element 6 is inserted into bin 1

Efficiency

Most combinatorial instances (known to be difficult)

- Few elements in each set variable
- Particularly efficient: the move often improves the solution when it is successful

Goal of the move: help LocalSolver get out of local minima (many set variables must be modified)
Tested on small random instances:

- Generated 50K instances/solutions
- Solutions with 10 set variables, 1 or 2 elements in each set variable
- Improvable solutions, but with no “obvious” improvements

⇒ Found **99.98%** improvements

Efficiency

Other instances

- Widens the gap between the set variables' weights when it is successful
- Easier to find improvements in the next iterations of the search

Assembly Line Balancing

- Apply the move to consecutive set variables to avoid violating precedence relations

Numerical results

Numerical results – 100 tasks Assembly Line Balancing instances

"large" benchmark from [1]

	LocalSolver 12.0		CP Optimizer 20.1.0		Gurobi 9.1	
	60s	600s	60s	600s	60s	600s
Nb, % feasible instances	525 100%	525 100%	525 100%	525 100%	459 87%	510 97%
Nb, % instances < 1% gap	487 93%	497 95%	447 85%	492 94%	326 62%	406 77%

Table 1: Numerical results – 100 tasks benchmark

[1] A. Otto, C. Otto, and A. Scholl. Systematic data generation and test design for solution algorithms on the example of salbpgen for assembly line balancing. European Journal of Operational Research, 228(1) :33–45, 2013.

Numerical results – 1000 tasks Assembly Line Balancing instances

"very large" benchmark from [1] – improvement of the literature's best known solution on 59% of the instances

	LocalSolver 12.0		CP Optimizer 20.1.0		Gurobi 9.1
	60s	600s	60s	600s	600s
Nb, % feasible instances	525 100%	525 100%	525 100%	525 100%	0 0%
Nb, % instances < 1% gap	500 95%	521 99%	310 59%	338 64%	0 0%
Avg gap	0.4%	0.1%	2.1%	1.7%	/

Table 2: Numerical results – 1000 tasks benchmark

[1] A. Otto, C. Otto, and A. Scholl. Systematic data generation and test design for solution algorithms on the example of salbpgen for assembly line balancing. European Journal of Operational Research, 228(1) :33–45, 2013.

LocalSolver

Numerical results – 1000 tasks Assembly Line Balancing instances

"very large" benchmark from [1] – improvement of the literature's best known solution on 59% of the instances

	LocalSolver 12.0		CP Optimizer 20.1.0		Gurobi 9.1	Moves deactivated	
	60s	600s	60s	600s	600s	60s	600s
Nb, % feasible instances	525 100%	525 100%	525 100%	525 100%	0 0%	525 100%	525 100%
Nb, % instances < 1% gap	500 95%	521 99%	310 59%	338 64%	0 0%	97 18%	209 40%
Avg gap	0.4%	0.1%	2.1%	1.7%	/	3.0%	1.9%

Table 3: Numerical results – 1000 tasks benchmark

[1] A. Otto, C. Otto, and A. Scholl. Systematic data generation and test design for solution algorithms on the example of salbpgen for assembly line balancing. European Journal of Operational Research, 228(1) :33–45, 2013.

Numerical results – 1000 tasks Assembly Line Balancing instances

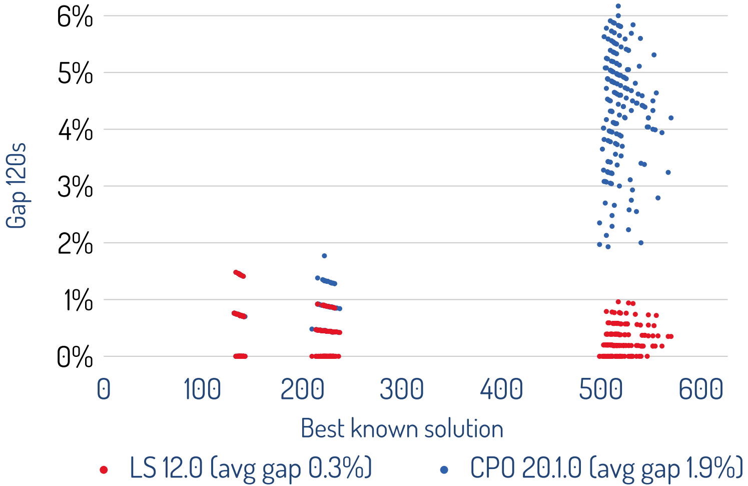


Figure 2: Gap to the best known solution in 120s

LocalSolver

Numerical results – Bin Packing instances

Performance improvements due to the move on LocalSolver 12.0 on the very hard Bin Packing instances from [2] :

- Gap to the best known lower bound : 0.44% → **0.36**% in 60s
- Improvements on **58**% of the 240 instances

[2] T. Gschwind and S. Irnich. Dual inequalities for stabilized column generation revisited. INFORMS Journal on Computing, 28(1) :175–194, 2016.

Conclusion

Conclusion

Local move based on ejection chains

- Applicable to any problem with a packing structure
- Helps LocalSolver get out of local minima
- Particularly efficient on combinatorial instances

Great performance improvements

- 0.4% gap on the Assembly Line Balancing Problem (60s)
- 0.36% gap on the Bin Packing Problem (60s)

Perspectives

Adapt our packing local move to apply it to more generalized packing problems

- Different set capacities
- Groups of elements
- Mandatory or forbidden assignments
- Bin-dependant element weights

Thank you for your attention