Solving the Assembly Line Balancing Problem with LocalSolver

Léa Blaise[1,2], Thierry Benoist[2] and Christian Artigues[1]

[1] LAAS-CNRS, Université de Toulouse, CNRS, INP, Toulouse, France
[2] LocalSolver, 24 Avenue Hoche, Paris, France
lblaise@localsolver.com

Keywords: local search, ejection chains, packing, solver.

1 Introduction and context

This paper introduces two algorithms implemented in LocalSolver, yielding great results on problems that present an ordering structure or a packing structure, such as the Assembly Line Balancing Problem.

LocalSolver is a global mathematical programming solver, whose goal is to offer a model-and-run approach to optimization problems, including combinatorial, continuous, and mixed problems, and to offer high quality solutions in short running times, even on large instances. It allows OR practitioners to focus on the modeling of the problem using a simple formalism, and then to defer its actual resolution to a solver based on efficient and reliable optimization techniques, including local search, but also linear, non-linear, and constraint programming. The local search algorithms implemented in LocalSolver are described in Gardi *et al.* (2014).

We focus here on the Assembly Line Balancing Problem, and we show how the algorithms implemented in LocalSolver produce excellent results on this problem. The Assembly Line Balancing Problem is described as follows. We consider a set of n tasks, of fixed duration, that are partially ordered by precedence relations. The problem consists in assigning the tasks to n workstations, while ensuring that the total duration of a workstation's tasks does not exceed the cycle time c. The precedence relations between the tasks impose that a task can be assigned to the same workstation as its predecessors, or to a later workstation, but cannot be assigned to an earlier workstation. The objective is to minimize the number of workstations used.

The LocalSolver model for the Assembly Line Balancing Problem is straightforward, written using set variables. In LocalSolver's formalism, a set variable of domain n is a decision variable whose value can be any (unordered) subset of $\{0, ..., n-1\}$. Each workstation S is then modeled using a set variable of domain n, whose value is equal to the set of tasks it contains. Since each task must be assigned to exactly one workstation, we constrain the set variables to form a partition of $\{0, ..., n-1\}$.

2 Greedy algorithm

In this Section, we introduce a greedy algorithm, working both as an initialization algorithm and as a "destroy and repair" local move, taking the capacity and precedence constraints into account. The implementation of this algorithm being as generic as possible, it is not only applicable to the Assembly Line Balancing Problem, but to any problem presenting an ordering structure and a packing structure on its set variables.

2.1 Building an initial feasible solution

In the LocalSolver model for the Assembly Line Balancing Problem, the precedence constraint between two tasks `t1` and `t2` is written:

```
constraint find(stations, t1) <= find(stations, t2);
```
(1)

The `find` operator takes two arguments: an array of set variables of domain n and an integer expression between 0 and $n-1$, and returns the index of the set variable which contains the requested element (or -1 if none of the set variables contain the element). The constraint written above then reads "The chosen workstation for task `t1` must be lower or equal to the chosen workstation for task `t2`".

When such precedence constraints are detected in the model, an ordering on the set variables can be deduced. We can then build an initial solution verifying the precedence constraints between the tasks, as well as the capacity constraints on the workstations. The initialization algorithm is as follows. Let $S_0, ..., S_{n-1}$ be the set variables of the model (in increasing order). Each set variable is initially empty. An element $0 \leq t \leq n-1$ (representing a task in the case of the Assembly Line Balancing Problem) is said to be eligible for insertion into a set variable S if all of its predecessors have already been assigned to a set variable, and if its weight (the task's duration) is lower than the remaining space in S (its capacity minus the total weight of the elements it already contains). Let E be the set of eligible elements. The first set to be filled is $S = S_0$. While the set of eligible elements is non empty, an element $t \in E$ is randomly chosen, and inserted into S. The set of eligible elements is then updated accordingly. When there is no eligible element anymore, we move on to the next set variable, until every element has been assigned to a set variable. The complexity of this algorithm is $O(n^2)$.

This algorithm allows LocalSolver to immediately obtain a first feasible solution for any instance of the Assembly Line Balancing Problem. This is particularly useful on the large instances, for which finding a feasible solution from a random assignment of the set variables' elements could take several seconds.

2.2 "Destroy and repair" local move

The greedy algorithm described in 2.1 can be adapted to be integrated into LocalSolver's local search component as a "destroy and repair" local move. Indeed, we can choose to apply it to fill a subset of the model's set variables rather than all of them. For example, in the case of the Assembly Line Balancing Problem, we can choose to apply it not to distribute the whole set of tasks into the different workstations, but to reorganize the tasks already assigned to a subset of workstations only. When applying this local move, the current solution is re-optimized by destroying and then rebuilding part of the solution.

To further diversify the explored solutions, the algorithm can either by applied by increasing order of the set variables (as in the initialization algorithm described in 2.1), or by decreasing order of the set variables.

3 Packing move based on ejection chains

In this Section, we describe another local move implemented inside LocalSolver's local search component, which makes use of the packing structure detected in the model (the total duration of a workstation's tasks must be lower than the cycle time). This local move is based on ejection chains: it consists in a series of element movements from one set variable to another. The move is similar to one of the algorithms described by Capua *et al.* (2018) to solve the Bin Packing Problem with Conflicts. The authors also describe a local

move based on ejection chains, with several differences. In their algorithm, all the sets and elements of the problem are considered, and a random ordering of the sets is imposed (an element can only be moved from its current set to another set of higher index). On the contrary, our local move only focuses on a subset of unordered set variables.

The local move's procedure is as follows. We start by choosing a random subset of the model's set variables. Let S be the set variable with the smallest weight among all chosen sets. An element $t \in S$ is randomly chosen to be ejected from S. The goal of the local move is to reorganize the other selected set variables' elements, so that t can be inserted into one of them. If t can be inserted into a new set variable $S^\star$ without violating the capacity constraint on $S^\star$, t is assigned to $S^\star$. The move is successful. If not, we try to swap it with another, strictly smaller element. In order to do this, we consider the smallest element $t' \notin S$ that can be replaced by t. If no such element exists, or if the weight of t' is larger than that of t, the local move results in failure, and we revert back to the previous solution. Otherwise, t' is ejected from its current set variable S', so that t can be inserted in its place. The same procedure is then repeated, with t' as the new current element, until the move ends in either success or failure. Since the weights of the ejected elements are strictly decreasing, the move ends in at most N steps, where N is the total number of elements in the selected subset of set variables. The complexity of each step is $O(N)$.

Figure 1 illustrates the local move's procedure on a small example with eight elements distributed into five set variables. The first element to be moved is 7, then 0, then 4, and finally 6.

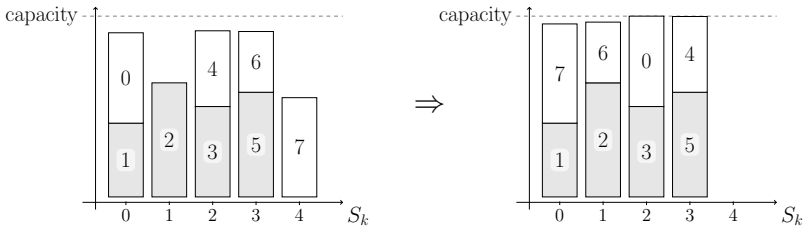

Fig. 1. Example – Solutions before (left) and after the local move (right)

This local move is particularly effective on the most combinatorial instances, in which each set variable contains very few elements. Indeed, if the smallest set variable selected contains only one element, and if the move is successful, the solution is improved (assuming that the number of used set variables is to be minimized). However, it is also useful on other types of instances, since it increases the weight gaps between the different set variables, which makes it easier to find improvements in the following iterations.

This local move can be applied to any problem presenting a packing structure. It improves LocalSolver's performance on various problems, such as the Bin Packing Problem, and the Assembly Line Balancing Problem.

4 Numerical results

We compare the performance of LocalSolver 10.5, CP Optimizer 20.1.0, and Gurobi 9.1, on the Assembly Line Balancing Problem, in 60 and 600 seconds of running time. The models we use to evaluate the performance of CP Optimizer and Gurobi are respectively given by Laborie (2020) and Pastor *et al.* (2007). We use the instances of the "very large" dataset proposed by Otto *et al.* (2013). The dataset includes 525 instances of 1000 tasks

to assign. The metric used is the gap to the best known solution, equal to the minimum between the best known solution given in (Otto *et al.* 2013) and the value yielded by LocalSolver after 10 minutes of running time. Indeed, LocalSolver improves the result given by the authors of the article on 59% of the instances. In Table 1, we give the percentage of feasible instances, the percentage of instances for which each solver reaches a solution within 1% of the best known solution, as well as the average gap.

Table 1. Comparison of LocalSolver's, CP Optimizer's, and Gurobi's performance – 1000 tasks

	LocalSolver		CP Optimizer		Gurobi	
	60s	600s	60s	600s	60s	600s
Feasible instances	100%	100%	100%	100%	0%	0%
Instances w. gap $< 1\%$	95%	99%	59%	64%	0%	0%
Average gap	0.4%	0.1%	2.1%	1.7%	–	–

We can see that LocalSolver's performance is significantly better than that of the other two solvers. The performance gap between LocalSolver and CP Optimizer is particularly striking on the most combinatorial instances, which are considered to be the most difficult.

The algorithms we describe here are however not dedicated to the Assembly Line Balancing problem, and can improve LocalSolver's performance on other problems as well. For example, the integration of the local move described in 3 into LocalSolver's local search enables it to get strictly better results on 58% of the very hard Bin Packing instances proposed by Gschwind and Irnich (2016), lowering the average gap to the best known lower bound from 0.44% to 0.36%.

5 Conclusion

In this paper, we considered a family of problems presenting a packing structure, such as the Bin Packing Problem, and an ordering structure, such as the Assembly Line Balancing Problem. We introduced a greedy algorithm, making use of both kinds of structures to build feasible solutions. We showed how it can be used both as an initialization algorithm and as a "destroy and repair" local move. We also introduced a packing move based on ejection chains, particularly efficient on the most combinatorial packing instances. Their integration into LocalSolver enables it to obtain great results on the targeted problems.

References

Gardi F., T. Benoist, J. Darlay, B. Estellon, and R. Megel, 2014, Mathematical Programming Solver Based on Local Search, Wiley.

Alena Otto and Christian Otto and Armin Scholl. Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing, 2013, *European Journal of Operational Research.*

Timo Gschwind and Stefan Irnich. Dual inequalities for stabilized column generation revisited, 2016, *INFORMS Journal of Computing.*

Renatha Capua, Yuri Frota, Luiz Satoru Ochi, and Thibaut Vidal. A study on exponential-size neighborhoods for the bin packing problem with conflicts, 2018, *Journal of Heuristics.*

Philippe Laborie. Solving the Simple Assembly Line Balancing Problem with CP Optimizer, 2020, https://www.linkedin.com/pulse/solving-simple-assembly-line-balancing-problem-cp-philippe-laborie/.

Pastor, Rafael and Ferrer, Laia and García, Alberto. Evaluating optimization models to solve SALBP, 2007, *Lecture Notes in Computer Science.*