Solution Repair by Inequality Network Propagation in LocalSolver

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

[1] LAAS-CNRS, Université de Toulouse, CNRS, INP, Toulouse, France
[2] LocalSolver, 36 Avenue Hoche, 75008, Paris, France
lblaise@localsolver.com

Keywords: constraint propagation, inequality networks, local search, solution repair, solver, disjunctive scheduling.

1 Introduction and context

This paper introduces a solution repair algorithm based on constraint propagation, designed to overcome the difficulties met by small neighborhood search on a family of tightly constrained problems.

LocalSolver is a 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).

This paper focuses on problems whose constraints include a network of two-variable inequalities. More precisely, it focuses on problems whose constraints comprise either linear inequalities between two variables or disjunctions of such inequalities. Many such problems arise in the field of scheduling: for example the Job Shop Problem (Fisher and Thompson 1963) and the Bridge Building Problem (Bartusch 1983), which are both characterized by generalized precedences and disjunctive resource constraints. However, this kind of structure is also typical of packing, layout, and mining problems.

These problems are highly constrained: in a good solution of a Job Shop or Bridge instance, the precedences and disjunctive resource constraints are often very tight. Because of that, moving from a solution of makespan x to a solution of makespan $x - 1$ requires a lot of small changes on many integer variables – the start times of the tasks. Being able to move from a good feasible solution to another using random small neighborhoods is then very unlikely: one would have to randomly target the right set of integer variables and to randomly shift them all by the right amount. For these reasons, the algorithms described in Gardi *et. al.* (2014) encounter serious difficulties on these problems. In the vast literature on job-shop scheduling by local search, these difficulties are overcome by exploiting higher level dedicated solution representations, such as the disjunctive graph (Vaessens *et. al.* 1994). In this work, we aim at keeping the modeling elements simple and we wish to target other problems as well. Hence, we focus on the direct integer variable representation.

2 Solution repair

The solution that we envisioned and implemented in LocalSolver to tackle this problem is a kind of constraint propagation: a promising but infeasible solution is gradually repaired one constraint at a time. This gradual procedure might remind one of the min-conflicts heuristic, used to repair an inconsistent assignment for the variables of a CSP, or of ejection chains algorithms, both recently studied in the context of scheduling problems, respectively in Ahmeti and Musliu (2018) and Ding *et. al.* (2019).

Before the search, specific constraints (such as generalized precedences and disjunctive resource constraints) are detected in the model and made into a constraint network. Those constraints will be the ones on which the repairing algorithm will apply when necessary. Let's consider any iteration of the local search, and let's assume that the current solution $\mathcal{S}_0$ is feasible. A local transformation is applied to the solution (for example, modification of a few start times), rendering it infeasible ($\mathcal{S}$). In the simplest case, the repair phase is similar to the classic constraint propagation of Constraint Programming (Rossi *et. al.* 2006), but has a few singularities. First, while constraint propagation generally aims at reducing the variables' domains, here the constraints are only propagated when they are violated and need repairing. Indeed, our only concern here is to build a feasible solution $\mathcal{S}'$ as close to $\mathcal{S}$ as possible. Then, we impose that, during the local move and along the successive constraint repairs, the variables must always be shifted in the same direction: if a variable's value was increased, it can still be increased further when repairing a constraint, but it can never be decreased, and reciprocally. This ensures that all the decisions taken during the current iteration (during the successive constraint repairs, and more importantly during the local move) are respected. This way, the successive repairs "follow the move's direction": they lead to finding a feasible solution as close as possible to $\mathcal{S}$, by amplifying the move rather than by cancelling it. At the end of the propagation, the algorithm was either able to repair all the constraints, and thus found a feasible solution $\mathcal{S}'$, or found a constraint that could not be repaired. In the latter case, all the changes are cancelled and the whole procedure starts over: the current solution is set back to $\mathcal{S}_0$, and a new move is applied.

2.1 Two-variable linear inequalities

We consider inequalities of the form

$$aX + bY \leq c$$

where X and Y are integer or floating point variables, and a, b, and c are any constants. When $a = 1$ and $b = -1$, these inequalities correspond to the generalized precedence constraints encountered in scheduling problems, but the following algorithm is not limited to this specific case.

Let's assume that the inequality $aX + bY \leq c$ is violated. Since the initial solution was feasible, at least X or Y has already been shifted in the "wrong" direction, and therefore cannot be shifted now in the repair direction. At most one variable (X by symmetry) can then be shifted in the repair direction: there exists only one way to repair the constraint, which consists in shifting X just enough ($X \leftarrow \frac{c-bY}{a}$). This is a necessary decision, as in every feasible solution following the move's choices, X is at least (resp. at most) $\frac{c-bY}{a}$.

When the only repairable constraints in the model are inequalities, the repair phase is equivalent to a particular kind of constraint propagation, which will be referred to as "half bound consistency" in the remainder of the paper. The reduction of a variable X's domain is propagated only if it excludes its current support x (only if the constraint is violated and needs repairing). After repairing the constraint, the new support of X is written x'. When propagating the reduction of X's domain, only one of its bounds is modified: either $x' > x$ (then X's value can no longer be decreased) and x' is its new lower bound, or $x' < x$ and x' is its new upper bound. Since each variable must always be shifted in the same direction, one of its bounds at most can be modified throughout the propagation.

An interesting property when the only repairable constraints in the model are inequalities is that, if there exists a feasible solution that respects the decisions of the local move, the algorithm is guaranteed to find one at the end of the propagation.

2.2 Disjunctions of two-variable linear inequalities

We consider disjunctions of inequalities of the form

$$\bigvee_i (a_i X_i + b_i Y_i \leq c_i)$$

where the X_i and Y_i are integer or floating point variables, and the a_i, b_i, and c_i are any constants. When $a_i = 1$ and $b_i = -1\ \forall i$, these disjunctions correspond to disjunctive resource constraints (disjunctions of size 2), or packing constraints in higher dimensions. However, the algorithm is not limited to this specific case here either.

Let's assume that a disjunction is violated. We will try to repair it in a non deterministic way. Since *a priori* none of the generalized precedences of the disjunction should prevail over the others, the algorithm chooses one at random and tries to repair it. If it cannot be repaired, it tries to repair the following one, and so forth. If none of them could be repaired, the propagation fails.

Let $aX + bY \leq c$ be the inequality that was randomly chosen for repair in the disjunction. If only one of its variables can be shifted in the repair direction, then the constraint is repaired as described in 2.1. It is also possible that both variables can be shifted in the repair direction, since the chosen inequality may not have been the one that was respected in the initial feasible solution $\mathcal{S}_0$. If so, the algorithm randomly chooses how to shift them. Let Δ be the distance to feasibility: $\Delta = aX + bY - c$, and let δ_X and δ_Y be the shares of the repair respectively attributed to X and Y, verifying $\delta_X + \delta_Y = \Delta$. There are four possible ways for the algorithm to repair the constraint: either X repairs it alone ($\delta_X = \Delta$), or Y repairs it alone ($\delta_Y = \Delta$), or X and Y equitably share the repair ($\delta_X = \delta_Y = \frac{\Delta}{2}$), or X and Y randomly share the repair ($\delta_X = random(1, \Delta - 1)$ and $\delta_Y = \Delta - \delta_X$).

Since the repair procedure of a disjunction is non deterministic, the previous properties, of half bound consistency and guarantee to find a feasible solution, do not hold anymore when such constraints are detected among the repairable constraints in the model. However, if there exists a solution that respects the decisions of the move, there is always a non-zero probability that the propagation leads to finding one, depending on whether the algorithm always takes the "right" random decisions when repairing a disjunction.

3 Application to scheduling problems

This method of solution repair by constraint propagation dramatically improves the performances of LocalSolver on problems with a network of two-variable linear inequalities.

3.1 Results on the Bridge Building Problem

The optimum value of the Bridge Building Problem is 104. Without our repair mechanism, within ten seconds of search, LocalSolver 9.0 is only able to find solutions of value 115 on average, and virtually never finds an optimum solution. But with the integrated repair mechanism, it always finds a solution of value 104 within four seconds of search, and very often finds one in less than one second.

3.2 Results on the Job Shop Problem

We compared the performances of LocalSolver 9.0 with and without this repair mechanism on three classic Job Shop instance classes: the FT class by Fisher and Thompson (1963), the LA class by Lawrence (1984), and the ORB class by Applegate and Cook (1991).

Table 1. Evolution of the gap between the average solution found by LocalSolver and the optimum solution, in 10 seconds and in 60 seconds, on different Job Shop instance classes

	Gap 10s		**Gap 60s**	
Instance class	**No repairs**	**With repairs**	**No repairs**	**With repairs**
FT	73%	7%	15%	3%
LA	246%	10%	91%	4%
ORB	120%	6%	22%	4%

As shown above in table 1, our repair mechanism enables LocalSolver not only to find very good solutions of many Job Shop instances, but also to find good solutions very quickly: less than 10% from the optimum within 10 seconds of search.

4 Conclusion

In this paper, we considered a family of problems, whose constraints comprise a network of two-variable linear inequalities. Although small neighborhood search algorithms may encounter serious difficulties on these problems, we introduced a solution repair algorithm based on constraint propagation, overcoming the difficulties met by small neighbourhood search. The two main specificities of our propagation algorithm are that a domain reduction is only propagated if it excludes the current support of the variable, and that each variable must always be shifted in the same direction. Its integration into LocalSolver dramatically improves its performances on the targeted problems, not only on classic scheduling problems such as the Job Shop Problem, but also on some 3D packing and mining industrial instances of our test base.

References

Ahmeti A. and N. Musliu, 2018, "Min-conflicts Heuristic for Multi-mode Resource-constrained Projects Scheduling", *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 237-244.

Applegate D., W. Cook, 1991, "A computational study of the job-shop scheduling problem", *ORSA Journal on Computing* 3, pp 149-156.

Bartusch M., 1983, "Optimierung von Netsplanen mit Anordnungsbeiziehungen bei knappen Betriebsmitteln", PhD thesis, Universitat Passau, Fakultat fur Matematik und Informatik.

Ding J., L. Shen, Z. Lü, and B. Peng, 2019, "Parallel machine scheduling with completion-time-based criteria and sequence-dependent deterioration", *Computers and Operations Research* (103), pp. 35-45.

Fisher H., G. Thompson, 1963, "Probabilistic learning combinations of local job-shop scheduling rules", *Industrial Scheduling*, Muth J., G. Thompson (Eds.), Prentice Hall, Englewood Cliffs, New Jersey, pp. 225-251.

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

Lawrence S., 1984, "Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement), Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburg, Pennsylvania.

Rossi F., P. Van Beek, and T. Walsh, 2006, "Handbook of Constraint Programming (Foundations of Artificial Intelligence)", Elsevier Science Inc., New York, NY, USA.

Vaessens, R. J. M., E. H. L. Aarts, and J. K. Lenstra, "Job shop scheduling by local search", *Informs Journal on computing*, 8(3), pp. 302-317.