

Solution Repair by Inequality Network Propagation in LocalSolver

Léa Blaise, Christian Artigues, Thierry Benoist

▶ **To cite this version:**

Léa Blaise, Christian Artigues, Thierry Benoist. Solution Repair by Inequality Network Propagation in LocalSolver. 16th International Conference on Parallel Problem Solving from Nature (PPSN 2020), Sep 2020, Leiden, Netherlands. 10.1007/978-3-030-58112-1_23 . hal-02866559

HAL Id: hal-02866559

https://laas.hal.science/hal-02866559

Submitted on 12 Jun 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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

June 2020

Abstract

This paper focuses on optimization problems whose constraints comprise a network of binary and ternary linear inequalities. These constraints are often encountered in the fields of scheduling, packing, layout, and mining. Alone, small-neighborhood local search algorithms encounter difficulties on these problems. Indeed, moving from a good solution to another requires small changes on many variables, due to the tight satisfaction of the constraints.

The solution we implemented in LocalSolver is a kind of constraint propagation: when the solution obtained after a local transformation is infeasible, we gradually repair it, one constraint at a time. In order to extend the local transformation rather than cancel it, we impose never to go back on the decision to increase or decrease the value of a variable. We show that the success of this repair procedure is guaranteed for a large class of constraints.

We apply this method to several scheduling problems, characterized by precedences and disjunctive resource constraints. We give numerical results on the Job Shop, Open Shop and Unit Commitment Problems, and show that our repair algorithm dramatically improves the performance of our local search algorithms.

Keywords: Constraint propagation – Local search – Repair algorithm – Solver – Disjunctive scheduling

1 Introduction

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 practitioners to focus on the modeling of the problem using a simple formalism, and then to defer its resolution to a solver based on efficient and reliable optimization techniques, including local search (but also linear, non-linear, and constraint programming). As described in [5] and [10], the local search components in LocalSolver are mostly based on small neighborhoods (flips, shifts, insertions, ...).

In this paper, we mainly target constraints defined by linear inequalities between two or three variables, and disjunctions and chains of such inequalities. Many scheduling problems comprise such a network of binary or ternary inequalities. For instance, the Job Shop Problem [9] is characterized by 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 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 [10] encounter serious difficulties on these problems. In

the vast literature on job-shop scheduling by local search (for example [4], [15] and [18]), these difficulties are overcome by exploiting higher level dedicated solution representations, such as the disjunctive graph. 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.

To tackle this problem, we designed a solution repair algorithm based on constraint propagation: a promising but infeasible solution is gradually repaired, one constraint at a time. This gradual procedure can be compared to ejection chains algorithms, originally proposed by Glover (1996) [11] to generate neighborhoods of compound moves for the Traveling Salesman Problem, and more recently studied by Ding *et al.* (2019) [6] in the context of scheduling problems. With powerful moves enhancing the local search, these methods yield great results. Our repair mechanism can also be compared to the min-conflicts heuristic introduced by Minton *et al.* (1992) [14] to solve CSPs, and more recently applied to the field of scheduling by Ahmeti and Musliu (2018) [1]. This method consists in creating a complete but inconsistent assignment for the variables of a CSP, and to repair the constraint violations until the assignment is consistent. The min-conflicts heuristic corresponds to selecting a variable involved in a violated constraint, and setting its value to the one minimizing the number of outstanding constraint violations.

This paper is organized as follows. Section 2 formally introduces our repair mechanism. Our method is simpler than the ones mentioned above, since we repair the violated constraints in the order we encounter them, by changing the values of the involved variables just enough to repair the current constraint. However, it yields satisfactory results in practice, has strong theoretical properties, and has the advantage of being very fast, which is crucial to integrate it in a high-performance solver. In Sections 3 and 4, we apply our method to binary constraints on Boolean and numeric variables respectively. We show that some constraints have very strong properties ensuring that the repair mechanism will succeed, and we characterize such constraints. In Section 5, we introduce some more complex constraints: disjunctions and chains, for which we adapt our repair mechanism. In Section 6, we introduce ternary constraints. Finally, we give numerical results in Section 7. We apply our method to the Job Shop, Open Shop, and Unit Commitment Problems, on which the repair procedure dramatically improves the performance of our local search algorithms.

2 Repair mechanism: definitions and general algorithm

In this section, we formally describe the repair mechanism implemented in LocalSolver to supplement its local search algorithms, which consists in gradually repairing an infeasible solution, one constraint at a time.

Definition 1 (Constraint). *A constraint is a relation on variables of an optimization problem that a solution must satisfy. It is characterized by its feasible set, which is a subset of the Cartesian product of the domains of its variables. The set of variables involved in a constraint $\mathcal{C}$ is denoted $var(\mathcal{C})$.*

In the remainder of the paper, when the context is clear, a constraint can refer either to its defining equation or to the related feasible set.

We consider any iteration of the local search, and we assume that this iteration starts from an initial feasible solution, denoted $\mathcal{S}_0$, in which each variable X has an initial value x_0, and an initial domain $\mathcal{D}_X = [\underline{x}, \overline{x}]$. A local transformation is applied to $\mathcal{S}_0$: we now have a solution $\mathcal{S}$, that we assume to be infeasible. The value of each variable X in the current solution is denoted x.

Property 1 (No backtracking). To ensure that the repair mechanism extends the local transformation rather than cancel it, we impose never to go back on a previous decision. In other words, if a variable has already been modified (in the local transformation, or when repairing a constraint), it can be modified again in the same direction, but it cannot be modified in the opposite direction. Then, a modification of a variable's value is equivalent to a domain reduction: when it increases (resp. decreases), its lower (resp. upper) bound is adjusted accordingly.

Because of Property 1, the repair phase is equivalent to a kind of constraint propagation, whose filtering algorithm will be referred to as "half bound consistency" (HBC) in the remainder

of the paper. Indeed, while the initial domain of each variable X is $\mathcal{D}_X = [\underline{x}, \overline{x}]$, it will be reduced by successive modifications of one of its bounds only throughout the propagation. The first modification of X's value x (in the local transformation, or during the propagation) determines which of X's bounds will have its modifications propagated. Therefore, throughout the iteration, a variable X's non empty domain is always of the form $[\underline{x}, b]$ or $[b, \overline{x}]$, with $b \in [\underline{x}, \overline{x}]$.

We now describe the repair procedure. It is also provided in pseudo-code form in Algorithm 1. After the local transformation is performed, all the constraints involving a variable that has just been modified (variables X such that $x \neq x_0$) are put in the propagation queue.

While the queue is non empty, we apply the HBC filtering algorithm on its front constraint $\mathcal{C}$, as follows. If $\mathcal{C}$ is already verified, it is skipped. Otherwise, we compute a new domain for each variable $x \in var(\mathcal{C})$, whose tighter bounds are chosen to ensure its consistency towards $\mathcal{C}$. If at least one of these reduced domains is empty, then $\mathcal{C}$ cannot be repaired: the propagation fails, and the constraints remaining in the propagation queue are ignored. Else, each variable $X \in var(\mathcal{C})$ is assigned a new value x'. If x is still in X's reduced domain, then $x' = x$. If not, x' is the value of the reduced domain that is closest to x: it is one of X's reduced bounds. Thus, x' is the projection of x on X's reduced domain.

We now consider the solution $\mathcal{S}'$, in which each variable $X \in var(\mathcal{C})$ takes the value x'. If this solution satisfies $\mathcal{C}$, the domain reductions are propagated: for each variable $X \in var(\mathcal{C})$ such that $x \neq x'$, its new bound x' is propagated (its other bound being left unchanged), and every other constraint involving X is added to the propagation queue. If $\mathcal{S}'$ violates the constraint, then there exists several ways to repair it, and several ways to choose a new valid value for the variables, none of which is supposedly better than the others. We detail how we deal with this situation in the following sections: we show that this situation never occurs for certain classes of constraints, and we explain how we randomly repair the more complex constraints.

When the queue is empty, either the propagation fails because there exists no feasible solution respecting the decisions of the local transformation, and the algorithm reverts back to its initial solution $\mathcal{S}_0$, or a feasible solution is found. In the latter case, for each variable, if its domain has been reduced, then each of these reductions corresponded to a modification of the same bound, and its current value is equal to this modified bound.

Algorithm 1 An iteration of local search

Require: initial feasible solution $\mathcal{S}_0$
  local transformation on $\mathcal{S}_0$: current solution $\mathcal{S}$
  $q \leftarrow \{\mathcal{C} : \exists X \in var(\mathcal{C}), x \neq x_0\}$
  while $q \neq \emptyset$ **do**
    $\mathcal{C} \leftarrow q.pop()$
    (We now apply the half bound consistency (HBC) filtering algorithm on $\mathcal{C}$)
    if $\mathcal{C}$ verified **then**
      continue
    end if
    $\forall X \in var(\mathcal{C})$, $\mathcal{D}'_X \leftarrow$ reduction of $\mathcal{D}_X$ regarding $\mathcal{C}$
    $\forall X \in var(\mathcal{C})$, $x' \leftarrow$ projection of x on $\mathcal{D}'_X$: solution $\mathcal{S}'$
    if $\exists X,\ \mathcal{D}'_X = \emptyset$ **then**
      $\mathcal{C}$ cannot be repaired, propagation fails: **break**
    else if $\mathcal{S}'$ infeasible **then**
      several ways to repair $\mathcal{C}$ (see Sections 3, 4, 5, 6 for precisions)
    else
      $q \leftarrow q \cup \{\mathcal{C}' \neq \mathcal{C} :\ \exists X \in var(\mathcal{C}) \cap var(\mathcal{C}'),\ x \neq x'\}$
      $\forall X \in var(\mathcal{C}),\ x' > x$ (resp. $x' < x$), update X's lower (resp. upper) bound x'
      $\mathcal{S} \leftarrow \mathcal{S}' :\ \forall X \in var(\mathcal{C}),\ x \leftarrow x'$
    end if
  end while
  if failed **then**
    revert to $\mathcal{S}_0$
  end if

3 Repair of binary constraints on Boolean variables

We first focus on the simple case of binary constraints on Boolean variables (*leq*, *nand*, *or*, *xor*):

$$X \leq Y; \quad X + Y \leq 1; \quad X + Y \geq 1; \quad X + Y = 1$$

When considering Boolean variables, Property 1 implies that each variable can only be modified once during each iteration of the local search. This implies that every time a violated constraint is propagated, there exists at most one way to repair it. Indeed, since the constraint is in the propagation queue, at least one of its variables must have been modified. Then, this variable cannot be modified again, and the constraint may only be repaired by changing the other one.

This procedure is similar to one iteration of the limited backtracking algorithm [8] for the 2-SAT Problem. We start from a solution where the variables that have already been modified by the local transformation have a definitive value, and the others only have a temporary value. When applying the HBC filtering algorithm on a violated constraint, two configurations arise. Either only one of the two Boolean variables already has a definitive value, and the other one is assigned the definitive value that repairs the constraint, or both variables already have a definitive value, and the propagation fails because the constraint cannot be repaired. The difference between the limited backtracking algorithm and ours is that we never take arbitrary decisions on which we can backtrack if we encounter a constraint that we cannot repair. Indeed, these arbitrary decisions were taken during the local transformation, which we choose not to reconsider.

Proposition 1. *If there exists a feasible solution that respects the decisions of the local transformation, the repair algorithm is guaranteed to find it.*

Sketch of proof. This ensues from the similarity between our repair algorithm on binary Boolean constraints and the limited backtracking algorithm. See [8]. □

Example 1. We consider a small problem with three Boolean variables X, Y and Z, and three binary Boolean constraints $\mathcal{C}_1 : X + Y \leq 1$, $\mathcal{C}_2 : Y + Z \geq 1$ and $\mathcal{C}_3 : X \geq Z$. We assume that the initial feasible solution $\mathcal{S}_0$ is such that $x_0 = 0$, $y_0 = 1$ and $z_0 = 0$. We assume that after the local transformation, the current solution $\mathcal{S}$ is infeasible and verifies $x = 1$, $y = y_0 = 1$ and $z = z_0 = 0$. The propagation takes place as follows:

- Modification of X (local transformation) $\Rightarrow q = \{\mathcal{C}_1, \mathcal{C}_3\}$.
- Propagate $\mathcal{C}_1 : X + Y \leq 1$. Repair: $y = 0$. Modification of $Y \Rightarrow q = \{\mathcal{C}_3, \mathcal{C}_2\}$.
- Propagate $\mathcal{C}_3 : X \geq Z$. Already verified $\Rightarrow q = \{\mathcal{C}_2\}$.
- Propagate $\mathcal{C}_2 : Y + Z \geq 1$. Repair: $z = 1$. Modification of $Z \Rightarrow q = \{\mathcal{C}_3\}$.
- Propagate $\mathcal{C}_3 : X \geq Z$. Already verified $\Rightarrow q = \emptyset$.

A feasible solution was found: $x = 1$, $y = 0$, $z = 1$.

4 Repair of binary constraints on numeric variables

We now consider binary constraints on numeric variables. First, we describe the specific binary constraints actually propagated in LocalSolver. We then characterize the more general form of binary constraints verifying useful properties.

4.1 Repair of binary linear inequalities on numeric variables

In addition to binary Boolean constraints, we consider inequalities of the form

$$aX + bY \leq c$$

where X and Y are integer or real variables, and a, b and c are any constants.

Remark 1. The special case where $a = 1$ and $b = -1$ corresponds to the generalized precedence constraints encountered in scheduling problems.

Repair mechanism. The propagation of these binary linear constraints follows the general repair algorithm described in Section 2. In more concrete terms, the application of the HBC filtering algorithm to a constraint $\mathcal{C}$: $aX + bY \leq c$ takes place as follows. If the inequality is already verified, its propagation is skipped. If the variables cannot be shifted enough in the right direction to repair $\mathcal{C}$, the propagation fails. Else, if only one of the variables can be shifted in the right direction (X by symmetry), the algorithm applies the only necessary and sufficient change on X to repair $\mathcal{C}$: $X \leftarrow \frac{c-by}{a}$. We show in Section 4.2 that the initial solution $\mathcal{S}_0$ being feasible ensures that the last possible case – both variables being able to move in the right direction – never actually happens.

4.2 Properties on binary linear inequalities

Property 2. If the initial solution $\mathcal{S}_0$ is feasible, then there exists at most one way to repair the constraint $\mathcal{C}$: $aX + bY \leq c$.

Proof. Let $aX + bY \leq c$ be a violated constraint. Then $ax + by > c$. Let us assume that there exists several ways to repair the constraint, which implies that both X and Y can be shifted in the repair direction. Therefore $ax \leq ax_0$, $by \leq by_0$, and $ax_0 + by_0 > c$: the constraint was also violated in the initial solution. □

Property 3. If the initial solution $\mathcal{S}_0$ is feasible, and if there exists a feasible solution compatible with the local transformation's decisions, then the algorithm is guaranteed to find it.

Sketch of proof. The proof comes from Property 2: whenever a violated constraint is encountered during the propagation, there exists exactly one necessary and sufficient way to repair it. Then, the algorithm can never take a "wrong" decision that would prevent it from finding a feasible solution at the end. □

4.3 Characterization of repairable binary constraints

In this section, we describe a condition on any binary constraint that ensures that, if there exists a feasible solution that respects the decisions of the local transformation, the propagation will succeed. Since this property ensures a high efficiency in the repair procedure, it is greatly desirable.

As seen in Section 4.2, this property is verified if and only if there always exists a necessary way to repair each encountered constraint. Indeed, if not, there is no indication as to which possible repair is the "right" one. As described in Section 2, when applying the HBC filtering algorithm on a constraint, we consider the solution in which the value of each variable X is the projection of its current value x on its reduced domain. If this solution is feasible, then it is a necessary repair.

We then introduce a condition on any binary constraint that ensures that if there exists a solution, then the projection of their current value on their reduced domains is a feasible solution.

Definition 2 (Minimal repair). *Let us consider a repair of an infeasible solution (x, y) into a feasible solution (x', y'). This repair is minimal if any intermediate solution $(\lambda x + (1-\lambda)x', \mu y + (1-\mu)y')$ with $\lambda, \mu \in]0, 1[$ is infeasible.*

Remark 2. A minimal repair is not always necessary. For example, two solutions (x, y_1) and (x, y_2) with $y_1 < y < y_2$ can both correspond to a minimal repair.

Definition 3 (Biconvex constraint). *A binary constraint $\mathcal{C}$ on real variables X and Y is called biconvex if*

$$\begin{cases} \forall x \in \mathcal{D}_X,\ \mathcal{C}^x = \{y \in \mathcal{D}_Y : (x, y) \in \mathcal{C}\} \text{ is convex} \\ \forall y \in \mathcal{D}_Y,\ \mathcal{C}^y = \{x \in \mathcal{D}_X : (x, y) \in \mathcal{C}\} \text{ is convex} \end{cases}$$

A binary constraint $\mathcal{C}$ on integer variables X and Y is called biconvex if

$$\begin{cases} (x_1, y_1) \text{ and } (x_2, y_1) \text{ feasible} \ \Rightarrow\ \forall \tilde{x} \in [x_1, x_2],\ (\tilde{x}, y_1) \text{ feasible} \\ (x_1, y_1) \text{ and } (x_1, y_2) \text{ feasible} \ \Rightarrow\ \forall \tilde{y} \in [y_1, y_2],\ (x_1, \tilde{y}) \text{ feasible} \end{cases}$$

Definition 4 (Path-connected constraint). *A binary constraint $\mathcal{C}$ on real variables X and Y is path-connected if for any two feasible solutions (x, y) and (x', y'), there exists a continuous feasible path between them. That is:*

$$\forall (x,y),(x',y') \in \mathcal{C},\ \exists f : [0,1] \to \mathcal{C} \text{ continuous}, f(0) = (x,y), f(1) = (x',y')$$

A binary constraint $\mathcal{C}$ on integer variables X and Y is path-connected if for any two feasible solutions (x, y) and (x', y'), there exists a path of feasible "neighbor"[1] solutions between them. That is:

$$\forall (x,y),(x',y') \in \mathcal{C},\ \exists \{f_0, ..., f_n\} \in \mathcal{C}, \begin{cases} f_0 = (x,y), f_n = (x',y') \\ \forall i < n,\ \|f_i - f_{i+1}\|_\infty \leq 1 \end{cases}$$

Lemma 1. *If a binary constraint $\mathcal{C}$ on integer or real variables X and Y is biconvex and path-connected, then for any two feasible solutions (x, y) and (x', y'), there exists a path of feasible solutions $(\tilde{x}, \tilde{y})$ such that $x \leq \tilde{x} \leq x'$ (resp. $x \geq \tilde{x} \geq x'$) and $y \leq \tilde{y} \leq y'$ (resp. $y \geq \tilde{y} \geq y'$).*

Such a path will be referred to as "bounded feasible path".

Proof. By symmetry, we assume $x \leq x'$. Let $\mathcal{D} = \{(\tilde{x}, \tilde{y}) : \tilde{x} \geq x\}$ and $\mathcal{C}' = \mathcal{C} \cap \mathcal{D}$. Since $\mathcal{C}$ is path-connected, for any two points A and B in $\mathcal{C}'$, there exists a path f from A to B in $\mathcal{C}$. Let us assume that f contains a point in $\mathcal{C} \setminus \mathcal{D}$. Then, A and B both being in $\mathcal{D}$, f crosses $\mathcal{D}$'s border an even number of time (and at least twice). Let (x, y_1) and $(x, y_2) \in \mathcal{C}'$ be the first and last of the intersection points. Since $\mathcal{C}$ is biconvex, $\forall \tilde{y} \in [y_1, y_2]$, the solution $(x, \tilde{y})$ is feasible. We can then define a new path f' from A to B, which is equal to f except between (x, y_1) and (x, y_2), where it is equal to this segment. f' is entirely in $\mathcal{C}'$.

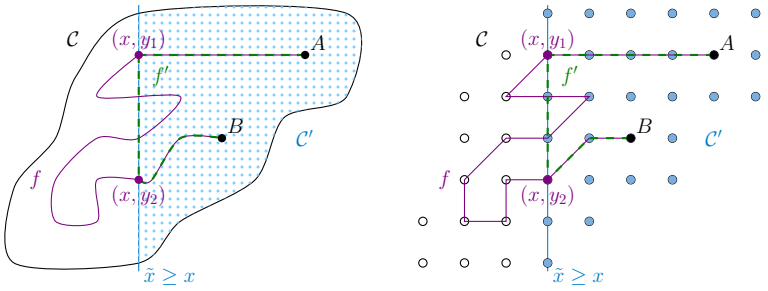


Figure 1: Bounded feasible path – real (left) and integer (right) variables

$\mathcal{C}'$ is then path-connected (Figure 1). By extension, the intersection of $\mathcal{C}$ and the rectangle formed by the two diagonal points (x, y) and (x', y') also is. □

Proposition 2. *If a binary constraint $\mathcal{C}$ on integer or real variables X and Y is biconvex and path-connected (and closed in the case of real variables), and if it is possible to repair it, then it admits a necessary repair. This repair consists in projecting the variables on their reduced domains.*

Proof. We assume that the initial solution (x_0, y_0) is feasible.

1. First, we assume that only X's value has changed: when the constraint $\mathcal{C}$ is propagated, the current solution (x, y_0) is infeasible (by symmetry, we assume $x > x_0$).

   (a) It is impossible to repair the constraint by shifting X only:
   If there exists $(x_1, y_0) \in \mathcal{C}$ with $x_1 > x$, then since $\mathcal{C}$ is biconvex, we also have $(x, y_0) \in \mathcal{C}$, which contradicts our initial assumption.

   (b) If there exists a feasible solution (x, y), then there exists exactly one minimal repair shifting only Y:
   Let us assume that there exists $y > y_0$ and $y' < y_0$ such that $(x, y) \in \mathcal{C}$ and $(x, y') \in \mathcal{C}$. Since $\mathcal{C}$ is biconvex, then $(x, y_0) \in \mathcal{C}$, which contradicts our initial assumption. Then, there exists at most one minimal repair shifting only Y.

[1] Two solutions (x_1, y_1) and (x_2, y_2) are neighbors when $\|(x_1, y_1) - (x_2, y_2)\|_\infty \leq 1 \Leftrightarrow |x_1 - x_2| \leq 1$ and $|y_1 - y_2| \leq 1$

If there exists a feasible solution (x, y) (with $y > y_0$, by symmetry), then $\mathcal{C} \cap \{(x, \tilde{y}) : y_0 < \tilde{y} \leq y\}$ being either closed (in the case of real variables) or finite (in the case of integer variables), there exists a minimal repair shifting only Y.

(c) If there exists a way to repair the constraint by shifting both variables, then there exists a minimal repair shifting only Y, which consists in projecting Y on its reduced domain:

Let us assume that there exists $x' > x$ and y' (by symmetry, $y' > y_0$ can be assumed) such that $(x', y') \in \mathcal{C}$. According to Lemma 1, there exists a bounded feasible path f from (x_0, y_0) to (x', y'), and f contains a point (x, y_1) with $y_0 < y_1 \leq y'$. Since $y_1 \leq y'$, there exists exactly one minimal repair increasing the value of Y. According to 1b, there exists exactly one minimal repair shifting only Y.

Paragraphs 1c and 1b prove that if there exists a feasible solution (x', y') with $y' > y_0$ (resp. $y' < y_0$), then $\forall \tilde{x} \geq x, \forall \tilde{y} \leq y_0$ (resp. $\forall \tilde{y} \geq y_0$), the solution $(\tilde{x}, \tilde{y})$ is infeasible. Therefore, there exists exactly one minimal repair $(x, y^\star)$, verifying $y^\star > y_0$, and $y^\star$ is the projection of Y on its reduced domain.

2. We now assume that both variables have been shifted: when the constraint is propagated, the current solution (x, y) is infeasible (by symmetry, we assume $x > x_0$ and $y > y_0$).

We show, as illustrated in Figure 2, that if it is possible to repair the constraint, then there exists a minimal repair shifting only one of the variables.

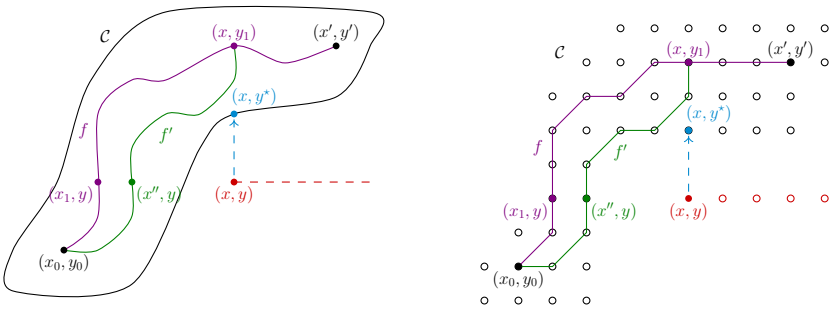


Figure 2: Necessary repair – real (left) and integer (right) variables

We assume that the constraint is repairable: there exists $x' \geq x$ and $y' \geq y$ such that $(x', y') \in \mathcal{C}$. According to Lemma 1, there exists a bounded feasible path f from (x_0, y_0) to (x', y'), and f contains two points (x, y_1) and (x_1, y), verifying $x_0 \leq x_1 \leq x'$ and $y_0 \leq y_1 \leq y'$. If $y_1 > y$, then, according to Lemma 1, there exists a bounded feasible path f' from (x_0, y_0) to (x, y_1), and f' contains a point (x'', y) such that $x_0 \leq x'' < x$. Since $\mathcal{C}$ is biconvex, $\forall \tilde{x} \geq x$, we have $(\tilde{x}, y) \notin \mathcal{C}$. Since (x_1, y) is feasible, we have $x_1 < x$. Likewise, if $x_1 > x$, then $y_1 < y$. By symmetry, let us assume $y_1 > y$: the feasible solution (x, y_1) respects the decisions of the local transformation, but (x_1, y) does not. Then, there exists exactly one minimal repair increasing the value of Y, and exactly one minimal repair shifting only Y.

As in 1, there exists exactly one minimal repair $(x, y^\star)$, verifying $y^\star > y_0$, and $y^\star$ is the projection of Y on its reduced domain. □

5 Repair of disjunctions and chains of binary linear inequalities on numeric variables

In addition to the previously mentioned constraints, we now 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 real variables, and the a_i, b_i and c_i are any constants, as well as chains of inequalities of the form

$$\bigwedge_i \left(aX_{L[f(i)]} + bX_{L[g(i)]} \leq c_{L[h(i)]}\right)$$

where X is an array of integer or real variables, L is a list variable (representing an ordering), and a, b and the elements of the array c are any constants.

Remark 3. When the a_i (resp. a) equal 1 and the b_i (resp. b) equal -1, these constraints describe packing or disjunctive resource constraints (tasks taken two by two in a disjunction, or ordered in regard of the list variable in a chain).

Example 2. Disjunctive resource constraints – non-overlapping of tasks scheduled on the same machine – are often encountered in the field of scheduling. When the number of tasks n on a resource is fixed, its disjunctive nature can be described by using $O(n^2)$ disjunctions of precedences:

$$\forall 1 \leq i < j \leq n,\ (S_j \geq S_i + d_i) \vee (S_i \geq S_j + d_j)$$

where S_i and d_i, respectively, denote the start time (variable) and the duration (fixed) of a task i. In this formulation, we consider that if two tasks i and j are scheduled on the same machine, either j starts after the end of i, or the opposite.

However, by using a chained constraint, we can reduce the number of constraints to $O(n)$. Indeed, in any feasible solution, the tasks are scheduled in a certain order. We can then formulate the constraint with a list variable L: the value of the list variable is a permutation, defining the order of the tasks.

$$\forall 1 \leq i < n,\ S_{L[i+1]} \geq S_{L[i]} + d_{L[i]}$$

In this formulation, we consider that the $(i+1)$-th task scheduled on the machine must start after the end of i-th task, creating a chained constraint of size $O(n)$.

Repair mechanism. When the constraints of the problem comprise disjunctions or chains, the properties listed in Section 4.2 do not hold anymore. Indeed, these constraints rarely admit a necessary repair. In order to efficiently repair them, we use a repair algorithm slightly different from Algorithm 1: the filtering algorithm we apply is based on random choices rather than on projections alone.

Repair of a disjunction. We assume that a disjunction is violated. Since *a priori* none of the inequalities 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 subsequent one, and so forth. If none of them can 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 Section 4.1. 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 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 $\delta_X + \delta_Y = \Delta$. The algorithm has four equiprobable ways 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 $\delta_Y = \Delta - \delta_X$).

Repair of a chain. Violated chains are repaired one index at a time. When considering one inequality of the chain, the repair procedure is very similar to that of an inequality of a disjunction.

Property 4. If there exists a solution that respects the decisions of the move, there is always a non-zero probability for the propagation to succeed, depending on whether the algorithm always takes the "right" random decisions.

Example 3 (Scheduling problem). We consider a scheduling problem with three tasks. Task t, of duration 3 and release date 1, and task t', of duration 2, must not overlap, and must both be scheduled before task t'', of duration 4. There are three integer variables S, S' and S'' (start times of the tasks), two precedences $\mathcal{P}_1$: $S - S'' \leq -3$ and $\mathcal{P}_2$: $S' - S'' \leq -2$, and one disjunctive resource constraint $\mathcal{R}$: $(S - S' \leq -3) \vee (S' - S \leq -2)$. We assume that the initial feasible solution $\mathcal{S}_0$ is such that $s_0 = 1$, $s'_0 = 4$ and $s''_0 = 6$. We assume that after the local transformation, the current solution $\mathcal{S}$ is infeasible and verifies $s = s_0 = 1$, $s' = s'_0 = 4$ and $s'' = 5$. The propagation can take place as follows:

- Modification of S'' (local transformation) $\Rightarrow q = \{\mathcal{P}_1, \mathcal{P}_2\}$.
- Propagation of $\mathcal{P}_1$: $S - S'' \leq -3$. Already verified $\Rightarrow q = \{\mathcal{P}_2\}$.
- Propagation of $\mathcal{P}_2$: $S' - S'' \leq -2$. Repair: $s' = 3$. Modif. of $S' \Rightarrow q = \{\mathcal{R}\}$.
- Propagation of $\mathcal{R}$: $(S - S' \leq -3) \vee (S' - S \leq -2)$. Repair (randomly chosen): $s = 2$, $s' = 0$. Modification of S and $S' \Rightarrow q = \{\mathcal{P}_1, \mathcal{P}_2\}$.
- Propagation of $\mathcal{P}_1$: $S - S'' \leq -3$. Already verified $\Rightarrow q = \{\mathcal{P}_2\}$.
- Propagation of $\mathcal{P}_2$: $S' - S'' \leq -3$. Already verified $\Rightarrow q = \emptyset$.

A feasible solution was found: $s = 2$, $s' = 0$, $s'' = 5$, as illustrated in Figure 3.

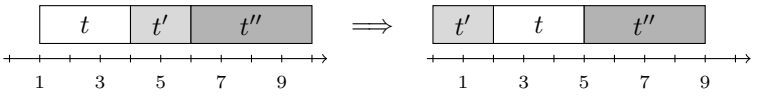


Figure 3: Initial and repaired solutions

6 Repair of ternary linear inequalities on numeric variables

We extend the repair mechanism to ternary linear inequalities, and to disjunctions and chains of such inequalities:

$$aX + bY + cZ \leq d$$
$$\bigvee_i (a_i X_i + b_i Y_i + c_i Z_i \leq d_i)$$
$$\bigwedge_i \left(aX_{L[f_1(i)]} + bX_{L[f_2(i)]} + cY_{L[f_3(i)]} \leq d_{L[f_4(i)]}\right)$$

Remark 4. If all as and cs equal 1 and all bs equal -1, these constraints describe precedences and disjunctive resource constraints on tasks of variable duration.

Repair mechanism. As in the previous section, when the current constraint is a ternary linear inequality, or a disjunction or chain of such inequalities, there likely exists several minimal ways to repair it. The filtering algorithm applied to these constraints is then non deterministic and based on random choices as well.

Let $aX + bY + cZ \leq d$ be a violated constraint. If only one or two variables can be shifted in the repair direction, the constraint is repaired like a binary linear inequality in a disjunction (see Section 5). If all three variables can be shifted in the repair direction, the algorithm chooses between eleven different repair methods. It can choose to repair the constraint by shifting one of the variables only, or by shifting two variables: either equitably or randomly. Finally, it can choose to repair it by shifting all three variables: either equitably or randomly.

The repair of disjunctions and chains of ternary inequalities is similar to that of disjunctions and chains of binary inequalities, described in Section 5.

7 Numerical results

7.1 Repair of binary constraints in shop scheduling problems

In this section, we consider the Job Shop and Open Shop Problems. In both problems, n jobs are divided into m activities each – one activity per machine. The machines are disjunctive, which

can be modeled using either $O(mn^2)$ disjunctions or $O(m)$ chains of size $O(n)$. In the Job Shop Problem, the activities of each job are ordered: there are $O(n^2)$ precedences, whereas in the Open Shop Problem, the jobs can be viewed as disjunctive resources. In both problems, the goal is to minimize the makespan.

In Table 1, we compare the performance of LocalSolver with and without our repair mechanism[1] on three classic Job Shop instance classes: the FT class by Fisher and Thompson [9], the LA class by Lawrence [13], and the ORB class by Applegate and Cook [3], as well as on a classic Open Shop instance class by Taillard [17]. We give the average optimality gap with and without repairs, after 10 and 60 seconds of search.

Table 1: Optimality gap – Job Shop and Open Shop Problems

Instance class	**Nb of instances**	**Gap 10s**		**Gap 60s**	
		No repairs	**Repairs**	**No repairs**	**Repairs**
Job Shop					
FT	3	73%	5%	15%	2%
LA	40	246%	8%	91%	3%
ORB	10	120%	6%	22%	3%
Open Shop					
4×4	10	41%	0.4%	41%	0%
5×5	10	65%	2.7%	65%	1.3%
7×7	10	102%	5.0%	93%	2.8%
10×10	10	569%	7.2%	261%	3.8%

While positive, it can be noted that these results remain behind those of dedicated scheduling algorithms or specialized methods (constraint-based scheduling, disjunctive graph-based local search) presented for example in [16] and [19] for the Job Shop Problem, and in [12] for the Open Shop Problem. However, as already mentioned, we are interested in more general forms of disjunctions, and we wish to keep our modeling elements simple and non-specialized.

On average, 77% of the improving moves on the Job Shop instances, and 62% on the Open Shop instances, gave a solution that was initially infeasible, but was successfully repaired by our algorithm.

7.2 Repair of ternary constraints in the Unit Commitment Problem

In this section, we focus on the Unit Commitment Problem, recently studied in [7]. We study a simplified version of the problem, where the level of production of each turned on unit is fixed to its average production rate. We model the problem as follows: each production range on any unit represents a task of variable duration. The constraints are chained disjunctive resource constraints: two consecutive tasks scheduled on a same unit must be separated by at least the unit's setup time. Since both the start time and the duration of each task are decision variables, the constraints are ternary.

In Table 2, we measure the performance improvement given by our repair algorithm in LocalSolver (compared to results obtained when turning repairs off), in 10 and 60 seconds. We also give the percentage of improving local transformations which needed repairing. We used the instances from [2]: the number of units varies from 10 to 100, and the number of time steps is 24.

[1] So far, we always assumed that the iteration's initial solution $\mathcal{S}_0$ was feasible, so as to guarantee certain desirable properties. Yet, the repair mechanism can also be ap- plied when $\mathcal{S}_0$ is infeasible, with a lower probability of success. Thus, it is also called in the early stages of LocalSolver's local search, before a feasible solution is found.

Table 2: Performance improvement – simplified Unit Commitment Problem

Nb units	**Nb instances**	**Improv. 10s**	**Improv. 60s**	**% repaired moves**
10	30	4%	3%	3%
20	27	7%	6%	3%
50	25	13%	10%	3%
75	19	12%	12%	4%
100	25	7%	14%	4%

8 Conclusion

In this paper, we considered a family of optimization problems, characterized by a network of binary and ternary linear inequalities. We introduced a solution repair algorithm based on constraint propagation, overcoming the difficulties met by small-neighborhood search algorithms. The two main specificities of our propagation algorithm are that a domain reduction is only propagated if it excludes the current value of the variable, and that each variable must always be shifted in the same direction. We also described some desirable properties on the constraints, which ensure the success of the repair procedure.

The main limitation of our approach is the possibility of failure on complex constraints repairs, which incurs the withdrawal of the tentative move. However, this is largely tempered in practice by the rapidity of the overall iteration process, which allows a huge number of moves to be tested in a short time frame, and generally leads to successful repairs. As a result, its integration into LocalSolver dramatically improves its performance on the targeted problems, not only on classic scheduling problems such as the Job Shop, Open Shop and Unit Commitment Problems, but also on some 3D packing and mining industrial instances.

References

[1] Ahmeti, A., Musliu, N.: Min-conflicts heuristic for multi-mode resource-constrained projects scheduling. In: Proceedings of the Genetic and Evolutionary Computation Conference. p. 237–244. GECCO '18, Association for Computing Machinery (2018)

[2] Angulo, A., Espinoza, D., Palma, R.: Thermal unit commitment instances for paper: A polyhedral-based approach applied to quadratic cost curves in the unit commitment problem. http://www.dii.uchile.cl/∼daespino/UC_instances _archivos/portada.htm

[3] Applegate, D., Cook, W.: A computational study of the job-shop scheduling problem. ORSA Journal on Computing **3**, 149–156 (1991)

[4] Beck, J., Feng, T., Watson, J.P.: Combining constraint programming and local search for job-shop scheduling. INFORMS Journal on Computing **23**, 1–14 (2011)

[5] Benoist, T., Estellon, B., Gardi, F., Megel, R., Nouioua, K.: Localsolver 1.x: A black-box local-search solver for 0-1 programming. 4OR **9**, 299–316 (2011)

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

[7] Dupin, N., Talbi, E.g.: Parallel matheuristics for the discrete unit commitment problem with min-stop ramping constraints. International Transactions in Operational Research **27**(1), 219–244 (2020)

[8] Even, S., Itai, A., Shamir, A.: On the complexity of timetable and multicommodity flow problems. SIAM J. Comput. **5**, 691–703 (1976)

[9] Fisher, H., Thompson, G.: Probabilistic learning combinations of local job-shop scheduling rules. Industrial Scheduling (1963)

[10] Gardi, F., Benoist, T., Darlay, J., Estellon, B., Megel, R.: Mathematical Programming Solver Based on Local Search. John Wiley & Sons, Ltd (2014)

[11] Glover, F.: Ejection chains, reference structures and alternating path methods for traveling salesman problems. Discrete Applied Mathematics **65**(1), 223 – 253 (1996)

[12] Grimes, D., Hebrard, E., Malapert, A.: Closing the open shop: Contradicting conventional wisdom. In: Gent, I.P. (ed.) Principles and Practice of Constraint Programming - CP 2009. pp. 400–408. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)

[13] Lawrence, S.: Resource-constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). Tech. rep., Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania (1984)

[14] Minton, S., Johnston, M.D., Philips, A.B., Laird, P.: Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. Artificial Intelligence **58**(1), 161 – 205 (1992)

[15] Peng, B., Lü, Z., Cheng, T.: A tabu search/path relinking algorithm to solve the job shop scheduling problem. Computers & Operations Research **53**, 154 – 164 (2015)

[16] Siala, M., Artigues, C., Hebrard, E.: Two clause learning approaches for disjunctive scheduling. In: Principles and Practice of Constraint Programming. pp. 393–402. Springer International Publishing (2015)

[17] Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64**(2), 278 – 285 (1993)

[18] Vaessens, R., Aarts, E., Lenstra, J.: Job shop scheduling by local search. Computing science notes, Eindhoven University of Technology (1994)

[19] Zhang, J., Ding, G., Zou, Y., Qin, S., Fu, J.: Review of job shop scheduling research and its new perspectives under industry 4.0. Journal of Intelligent Manufacturing (2017)