Explanation-Guided Large Neighborhood Search

Charles Prud'homme[1], Xavier Lorca[1], and Narendra Jussien[1]

École des Mines de Nantes, INRIA TASC, LINA UMR CNRS 6241,
FR-44307 Nantes Cedex 3, France
{Charles.Prudhomme,Xavier.Lorca,Narendra.Jussien}@mines-nantes.fr

Abstract. One of the most well-known and widely used local search techniques for solving optimization problems in Constraint Programming is the Large Neighborhood Search (LNS) algorithm. Such an hybrid technique is, by nature, very flexible and can be easily integrated within standard backtrack procedures. One of its drawbacks is that the relaxation process is quite often problem dependent and requires randomization to ensure search diversification. Several works have been dedicated to overcome this issue through problem independent parameters. In this paper, we show that LNS hard-to-find problem independent relaxation strategy can be addressed using an explanation-based search. We evaluate our proposal on a set of optimization problems. We show that our approach is at least competitive with or even better than state-of-the-art algorithms paving the way to a new use of explanation-based approaches for improving search.

1 Introduction

In the context of constraint programming (CP) for optimization problems, one of the most well-known and widely used local search techniques is the Large Neighborhood Search (LNS) algorithm[11]. A LNS is a two-phase algorithm which improves a given solution by alternatively destroying and repairing it. LNS combines Local Search by relaxing part of a solution and Constraint Programming for repairing process and bounding the objective variable. This technique has proven to be very flexible and to be easily integrated within standard backtrack procedures. One drawback of LNS is that the relaxation process is quite problem dependent [1, 2, 7]. A random selection of the variables to unfix may be considered first. But problem dedicated variations tends to be more efficient[1, 2, 7, 11]. More sophisticated neighborhoods, which did not rely on a dedicated problem, have been proposed in [8]. They introduced a Propagation-Guided LNS: the volume of domain reduction related to propagation enables to link variables together inside or outside neighborhoods. Such approaches rely on a parametrization of the heuristics.

The general behavior of LNS is described in Algorithm 1. Starting from an initial solution S, LNS selects and releases a subset of variables, referred to as the *neighborhood* (line 3). The partially destroyed solution is then repaired in order to find an improving solution S' (line 4). If such a solution is found (line 5),

Algorithm 1 Large Neighborhood Search

```
Require: an initial solution S
1: procedure LNS
2:     while Optimal solution not found and a stop criterion is not encountered do
3:         RELAX(S)
4:         S' ← FINDSOLUTION()
5:         if EVAL(S') < EVAL(S) then
6:             S = S'
7:         end if
8:     end while
9: end procedure
```

it is stored (line 6). These operations are executed until the optimal solution is found or a stopping criterion (for instance a time limit) is encountered (line 2).

During the last decade, explanation-based techniques have regained attention in CP. Nogoods and explanations have long been used in various paradigms for improving search [3, 4, 9, 10]. An explanation records some sufficient information to justify an inference made by the solver (domain reduction, contradiction, etc.). It is made of a set of propagators (a subset of the set of the original propagators of the problem) and a set of decisions taken during search.

Explanations have been successfully used for improving constraint programming solving. Both complete (as the `mac-dbt` algorithm [5]) and incomplete (as the `decision-repair` algorithm [4]) techniques have been proposed. Those techniques follow a similar pattern: learning from failures by recording each domain modification with its associated explanation (provided by the solver) and taking advantage of the information gathered to be able to react upon failure by directly pointing to relevant decisions to be undone.

Despite being possibly very efficient, explanations suffer from several drawbacks: they naturally induce several costs (memory, cpu) and their implementation can be quite intrusive. Moreover, explanations, as constraint programing itself, is inclined to satisfaction problems rather than optimization problems.

In this paper, we show that LNS hard-to-find problem independent technique can be addressed using explanations. A first contribution relies on a configuration-free neighborhood selection based on the explanation of the current solution. A second contribution is the operational implementation of this mechanism. We evaluate our proposal on a set of optimization problems. We show that our approach is competitive with or even better than state-of-the-art algorithms paving the way to a new use of explanation-based approaches for improving search.

2 The Explanation-Guided LNS

Explanation-Guided LNS (EGLNS), an explanation-guided neighborhood for Large Neighborhood Search is a generic, configuration-free approach to choose variables to relax, based on the explanation of the non optimal nature of the current solution. Basically, we modify the RELAX(S) method of Algorithm 1. For sake of simplicity, the following descriptions are stated in a minimization con-

Algorithm 2 Explanation-Guided neighborhood (in a minimization context)

```
Require: o: the objective variable
1: procedure RELAX(S)
2:     if a new solution has been found then
3:         computeNeighborhood(S, o)
4:     else
5:         diversify()
6:     end if
7: end procedure
```

```
1: procedure computeNeighborhood(S, o)
2:     k ← 0, I_k ← {Low_o}
3:     P_k ← relaxPath(pathTo(S), o, I_k)
4:     while apply(P_k) fails do
5:         k ← k + 1
6:         I_k ← I_{k−1} ∪ {Low_o + k}
7:         P_k ← relaxPath(P_{k−1}, o, I_k)
8:     end while
9: end procedure
```

text. Obviously, they can be adapted to a maximization context with marginal modifications. Algorithm 2 is decomposed in two main steps: (a) a new solution has been found, a new neighborhood has to be computed (left hand side, line 2-3); (b) or the current neighborhood does not let to find a new solution, it has to be diversified (left hand side, line 5).

We introduce the intuition of the method computeNeighborhood of Algorithm 2 that builds such a neighborhood (right hand side, lines 1-9). Given a solution S and its decision path P (*i.e.*, a chronologically ordered sequence of decisions), retrieving the subset of decisions of P which are related to the restrictions of Dom_o, the initial domain of the objective variable o, provides enough information to define a neighborhood. Removing such decisions from P relaxes the domain of the objective variable o, and enables to improve the value assigned to o, *i.e.*, a value in $[Low_o, o^*[$, where o^* is the value assigned to o in the last solution. Computing decisions related to the value removals is achieved thanks to the computation of explanations.

The computation of the neighborhood is conditioned by the discovery of a new solution S. First, an interval I_0 is set to the initial domain lower bound of the objective value (line 2). Then, an initial neighborhood is computed by a call to the relaxPath method (line 3) (not described here): it consists in a relaxation of the decision path which led to S (retrieved thanks to the pathTo(S) method). Decisions related to the removal of I_0 from Dom_o are removed from the decision path. The newly updated decision path P_k is applied (line 4), with respect to the chronological order. Due to the cut on the objective variable o, the application of P_k may fail. In that case, the neighborhood needs to be updated until P_k can safely be applied. A new interval I_k is computed by adding the k^{th} value after Low_o, contained in Dom_o (line 5-6). Building a larger interval relaxes P_k a little more at each step k.

When the relaxed decision path has been applied without fail but no improving solution has been found during the exploration of the remaining search space, a diversification of the neighborhood is required to explore another search space. This is achieved through a call to the method diversify (left hand side, line 5). We choose to randomly select a set of variables to fix, based on S. To ensure that such method ends, the number of variables selected monotonically decreases with failures.

3 Evaluation

The central issue of the Explanation-Guided LNS algorithm is to speed up the LNS process by designing neighborhoods directly related to the objective variable and yet to explore more appropriate parts of the search space. This section demonstrates the benefits of plugging Explanation-Guided LNS in and its robustness.

Benchmark protocol. Random LNS, Propagation-Guided LNS and Explanation-Guided LNS were implemented in Choco-13.03 [12], a Java library for constraint programming. All the experiments were done on a Macbook Pro with a 6-core Intel Xeon at 2.93Ghz running on MacOS 10.6.8, and Java 1.7. Each run had a five minutes time limit. EGLNS has been evaluated on three problems extracted from the MiniZinc 1.6 [6] distribution. Optimization problems without any global constraints have been selected for the evaluations.[1] Moreover, we kept instances for which classic backtrack algorithm finds at least one solution within a five minutes time limit (a LNS needs an initial solution to be activated). Each example is run ten times and the arithmetic mean of computation time, best objective and number of solutions are reported. Note that the first solution of each run is always the same one, whatever versions of LNS is plugged in. The results are represented with tables. Tables are presented to report pairwise contribution of approaches in the best found solutions. The column *worse* (resp. *same* and *better*) counts the number of instances where a given approach provides a worse (resp. equivalent and better) objective value than the best one found with another approach in the time limit.

3.1 Comparisons with RLNS and PGLNS

The main motivation of this paper is to improve the resolution of optimization problems. In this section, we compare EGLNS with a Random LNS and Propagation-Guided LNS. Even though the former is a naive approach, it comes with two benefits: it does not require any configuration and, the strong diversification it provides kicks it out of local optimum. The latter, neither custom designed neighborhood but relying on global parameters, has been proven to be very efficient.

Explanation-Guided LNS versus Random LNS. As a more complex approach but still configuration-free, EGLNS may be less efficient in getting out of local optima and may suffer from the computation of the explanations. The first evaluation concerns the contribution of EGLNS in comparison with RLNS. Table 1 shows that EGLNS finds an equivalent or better solution in 82% of the instances within the time limit. And the gain of using EGLNS (144 instances) clearly overcomes the loss (43 instances). Thanks to the explanations, EGLNS discovers the structure of the problem and forces first to focus on it. In that sense, EGLNS is

[1] Global constraints require implementation of specific explanation schemas.

Table 1. Impact of EGLNS in comparison with RLNS.

Problem	inst.	worse	same	better
`cutting stock`	100	26	9	**65**
`prize collect.`	66	10	**43**	13
`vrp`	74	7	1	**66**
total	240	43	53	**144**

Table 2. Impact of EGLNS in comparison with PGLNS.

Problem	inst.	worse	same	better
`cutting stock`	100	41	9	**50**
`prize collect.`	66	13	**42**	11
`vrp`	74	26	1	**47**
total	240	80	52	**108**

a kind of an instance dedicated approach. The difficulty of the Prize Collecting Problem remains more on the proof of the optimum value rather than on finding it. That explains why there are so many equivalent solutions between EGLNS and RLNS.

Explanation-Guided LNS versus Propagation-Guided LNS. Another generic approach outperforms RLNS, even if it requires parametrization.[2] The next evaluation, in Table 2, compares EGLNS with PGLNS. Once again, even though it is less obvious than with RLNS, selecting the neighborhoods with EGLNS is a good choice: it finds an equivalent or better solution in 66.6% within the time limit. The gain of using EGLNS (108 instances) overcomes the loss (80 instances) once again. The closeness relation[3] in PGLNS is naturally induced by EGLNS thanks to both the explanation network and the use of the decisions (which impact variables through the propagation phase). However, PGLNS does not focus on a particular subpart of the problem (due to randomness), whereas EGLNS always starts from the objective variable to explore the subparts of the problem.

We show that the Explanation-Guided LNS is competitive with both the Random LNS and the Propagation-Guided LNS, two problem-independent algorithms. The standard deviations of the best objective for RLNS, PGLNS and EGLNS are respectively 2.4%, 2.1% and 2.6%, which make them very stable. However, the standard deviation of the computation time are respectively 73%, 61% and 54% in average. Such observations distinct the robustness of EGLNS.

4 Conclusion and future work

In this paper, we propose a configuration-free approach to compute neighborhoods in a LNS, based on the explanation of the non optimal nature of the current solution. We show that our approach is competitive with or even better than state-of-the-art algorithms, on a set of optimization problems. Our results

[2] We use the default parameters of [7]: a list of size 10, a constant valued to 30, a dynamic epsilon and a fair combination of PGLNS, reverse PGLNS and random.

[3] The closeness relation, defined in [7], enables to "discover the closely linked subparts of the problem (...) by following the effects of the propagation".

are encouraging and should be validated on a larger set of problems. Both the search strategy (how decisions are selected) and the propagation engine (how events are relayed in the constraint network) influence the quality of the computed explanations. Since the explanation of a domain reduction is not unique, our approach would benefit from more concise explanations, *e.g.*, by enabling global constraints. Future works should focus on the diversification procedure, more particularly exploiting explanation of the conflict induced by a cut, and NoGoods should be the key combination to avoid multiple exploration of nested search spaces.

References

1. Alain Chabrier, Emilie Danna, Claude Le Pape, and Laurent Perron. Solving a network design problem. *Annals of Operations Research*, 130(1-4):217–239, 2004.
2. Emilie Danna and Laurent Perron. Structured vs. unstructured large neighborhood search: A case study on job-shop scheduling problems with earliness and tardiness costs. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming CP 2003*, volume 2833 of *Lecture Notes in Computer Science*, pages 817–821. Springer Berlin Heidelberg, 2003.
3. M. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.
4. N. Jussien and O. Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1):21–45, July 2002.
5. Narendra Jussien, Romuald Debruyne, and Patrice Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Principles and Practice of Constraint Programming (CP 2000)*, number 1894 in Lecture Notes in Computer Science, pages 249–261, Singapore, September 2000. Springer-Verlag.
6. Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *In: Proc. of 13th International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.
7. Laurent Perron and Paul Shaw. Combining forces to solve the car sequencing problem. In Jean-Charles Régin and Michel Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 3011 of *Lecture Notes in Computer Science*, pages 225–239. Springer Berlin Heidelberg, 2004.
8. Laurent Perron, Paul Shaw, and Vincent Furnon. Propagation guided large neighborhood search. In *CP'04*, pages 468–481, 2004.
9. P. Prosser. MAC-CBJ: maintaining arc consistency with conflict-directed backjumping. Technical Report Research Report/95/177, Dept. of Computer Science, University of Strathclyde, 1995.
10. T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problem. *IJAIT*, 3(2):187–207, 1994.
11. Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In Michael Maher and Jean-Francois Puget, editors, *Principles and Practice of Constraint Programming CP98*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer Berlin Heidelberg, 1998.
12. Choco team. *choco-13.03*. http://www.emn.fr/z-info/choco-solver/index.php?page=choco-3, 2013.