Cost Impact Guided LNS

Michele Lombardi[1] and Pierre Schaus[2]

[1] DISI, University of Bologna
michele.lombardi2@unibo.it
[2] ICTEAM, Université Catholique de Louvain, Belgium
pierre.schaus@uclouvain.be

Abstract. In Large Neighborhood Search (LNS) [14], a problem is solved by repeatedly exploring (via tree search) a neighborhood of an incumbent solution. Whenever an improving solution is found, this replaces the current incumbent. LNS can improve dramatically the scalability of CP on large real world problems, provided a good neighborhood selection heuristic is available. Unfortunately, designing a neighborhood heuristic for LNS is still largely an art and on many problems beating a random selection requires a considerable amount of both cleverness and domain knowledge. Recently, some authors have advocated the idea to include in the neighborhood the variables that are most directly affecting the cost of the current solution. The proposed approaches, however, are either domain dependent or require non-trivial solver modifications. In this paper, we rely on constraint propagation and basic solver support to design a set of simple, cost based, domain independent neighborhood selection heuristics. Those techniques are applied on Steel Mill Slab problems illustrating the superiority of some of them over pure random relaxations.

1 Introduction

Large Neighborhood Search (LNS) is a powerful hybrid method that employs ideas from Local Search to dramatically improve the scalability of Constraint Programming (CP) on large scale optimization problems. Specifically, LNS is an iterative approach that starts from an incumbent solution and tries to improve it by using CP to explore a neighborhood. This neighborhood is usually defined by *freezing* a subset of variables, which are left assigned to the value they had in the incumbent solution. The remaining variables are instead *relaxed*, meaning that their domain is restored to its initial content. Typically, the neighborhood is explored under some search limit (e.g. maximum number of backtracks or time), to avoid spending too much time in a single iteration. If an improving solution is found, it becomes the new incumbent.

Formally, let $P = \langle z, X, D, C \rangle$ be a Constraint Optimization Problem, where X is the set of variables, D is the set of the variable domains (with D_i being the domain of x_i), and z is a cost variable. Without loss of generality we assume that z has initial domain $[-\infty, \infty]$ and must be minimized. The set C contains the problem constraints. Each constraint c_i is defined over a subset of variables $S(c_i)$, known as its *scope*. The scope can include both the X and the z variables.

We view a (partial) assignment τ as a particular constraint that forces each variable x_i in its scope $S(\tau)$ to assume a specific value $v_i = \tau(x_i)$. An assignment τ is a solution if $S(\tau) = X$ and the problem $P_\tau = \langle z, X, D, C \cup \tau \rangle$ is consistent (none of the constraints detected an infeasibility). We use the special notation σ to refer to solutions.

In each LNS iteration we start from a solution σ, then we select a subset of variables to relax X_R and we build a partial assignment τ such that:

- the scope includes the variables to freeze, i.e. $S(\tau) = X \setminus X_R$
- $\tau(x_i) = \sigma(x_i)$ for each $x_i \in S(\tau)$

then we try to find a solution for P_τ with an improving cost. More precisely, let the notation $lb_\tau(x_i)$ refer to the lower bound of the domain of x_i after the propagation has reached a fix point on P_τ. Similarly, we use the notation $ub_\tau(x_i)$ for the upper bound. Then a solution σ' has better cost than σ iff $lb_{\sigma'}(z) < lb_\sigma(z)$.

The choice of the variables X_R to relax is crucial for the effectiveness of the approach. Currently, most of the best selection heuristics are domain specific and require a great deal of both expertise and knowledge to be formulated. While several researchers have addressed the topic of independent black-box search for CP [11, 15, 8, 2], much less effort has been dedicated to make neighborhood selection in LNS problem independent. Some of the most relevant attempts in this direction are summarized in Section 2.

In this paper, we propose a novel cost driven, domain independent neighborhood selection method, based on the information collected by the progressive re-application of the current incumbent solution (a.k.a. a *dive*). The method is described in Section 3. In Section 4 we report results for the Steel Mill Slabs problem and in Section 5 we offer some concluding remarks.

2 Related work

This section describes existing domain independent and cost based approaches for neighborhood selection in LNS. The discussion does not include *adaptive* approaches, where the goal is to automatically learn the best neighborhood (from a given pool) or the best parameters for a selection method. The interested reader is invited to check [5, 3, 12, 6] for more details. As a remark, the integration of domain independent neighborhood selection and adaptive schemes offers a lot of opportunities and represents a very interesting topic for future research.

2.1 Propagation Guided LNS (PGLNS)

This approach is introduced in [10], where the authors define two neighborhood selection methods relying on information coming from constraint propagation. The first method defines the set of variable to *freeze* by incrementally building a partial assignment τ, starting from an empty scope. In particular:

1. First, a variable is selected at random from X and inserted in $S(\tau)$

2. The fix point for P_τ is reached
3. Then the next variable is selected at random among the 10 with the largest (non-zero) impact (defined as in [11]). If the list is empty the selection is done at random from $X \setminus S(\tau)$.
4. The process goes back to step 2, until the size of the search space of P_τ (actually, an approximation of that) is small enough.

The underlying idea is that of *freezing* related variables.

The second method, name *Reversed* PGLNS also follows an incremental scheme, but performs no propagation and relies on the availability of a closeness measure between pairs of variables. The method builds incrementally the set X_R of variables to *relax* by always choosing the next variable among the 10 with the largest (non-zero) closeness to the ones in X_R. The choice is made at random if the list is empty. As one can see, this approach is based on the idea of *relaxing* related variables. In their implementation, the authors interleave PGLNS and Reversed PGLNS and use the impacts from PGLNS to obtain the closeness scores. The Reversed PGLNS approach performed best in their experimentation.

2.2 Cost Based Neighborhoods for Scheduling Problems

As a major drawback, the PGLNS approach makes no effort to exploit the connection between variable assignment and the cost variable z. In [3], the authors propose a cost driven neighborhood selection method for scheduling problems. The main underlying idea is to include in the set X_R the variables that are most directly affecting the cost of the current solution. The authors successfully apply this idea to Job-Shop Scheduling by choosing the start variables to be relaxed among those having the smallest slack. Activities with a larger slack start to be considered only after a certain number of non-improving LNS iterations. Unfortunately, this approach cannot be considered really problem independent.

2.3 Generic adaptive heuristics for LNS

Several cost driven and domain independent neighborhood selection methods are proposed in [7], the most successful ones being based on the so-called *dynamic impact* of a variable. The dynamic impact tries to capture the effect that relaxing a variable would have on the problem cost. Specifically, let σ be the incumbent solution and let $\tau_{i,v}$ be an assignment that is identical to σ, except that $\tau_{i,v}(x_i) = v$. Then the impact of the pair (x_i, v) is defined as:

$$\mathcal{I}^d(x_i, v, \sigma) = lb_{\tau_{i,v}}(z) - lb_\sigma(z) \tag{1}$$

Note that $\tau_{i,v}$ is not guaranteed to be a solution, since it may make the problem infeasible. To avoid this problem, during the impact evaluation the authors disregard all constraints that are not needed for the cost computation.

The authors obtain their best results by selecting the variables to be relaxed with a probability proportional to their *mean dynamic impact*, defined as:

$$\overline{\mathcal{I}}^d(x_i, \sigma) = \frac{1}{|D_i|} \sum_{\substack{v \in D_i, \\ v \neq \sigma(x_i)}} \mathcal{I}^d(x_i, v, \sigma) \tag{2}$$

which requires to compute the dynamic impact for each value in the original domain D_i. The author evaluated the neighborhood selection heuristic by performing a single LNS iteration starting from several reference solutions. The method based on mean impact was able to improve the solution more frequently than a random selection. This evaluation approach, although sound, may be biased by the choice of the reference solutions (e.g. improving a solution with loose constraints is very different improving one with tight constraints). The described approach has furthermore some drawbacks:

- Ignoring problem constraints (except for those needed for the cost computation) does not account for the indirect cost impact that a variable may have due to other constraints.
- Automatically detecting the constraints needed for the cost computation may not be doable. In such situation, the user would need to manually specify them, requiring a custom extension in the modeling interface.
- It is not necessarily true that measuring the cost impact of a variable under the assumption that it is the last to be assigned leads to a reliable evaluation.

3 Cost Impact Guided LNS

In this work we extend the idea introduced in [3] that an effective LNS neighborhood heuristics should be cost based. Our goal is to make this principle independent of the problem, by relying on the propagation over the cost variable, similarly to [7]. At the same time, however, we wish to avoid the drawbacks that we have identified in the previous section.

Our method relies on a cost impact metric based on the variation of the lower bounds of the cost variable[1]. Unlike [7], however, we collect those variations by incrementally re-applying the current solution in rearranged order, i.e. by performing a *dive*.

Specifically:

Definition 1. *Let π be a permutation of the variables in X and let k be the position of x_i in π. Then the* cost impact *of x_i w.r.t. a solution σ is the quantity:*

$$\mathcal{I}^z(x_i, \sigma, \pi) = lb_{\tau_{\pi,k}}(z) - lb_{\tau_{\pi,k-1}}(z) \tag{3}$$

where:

$$S(\tau_{\pi,k}) = \{x_{\pi_j} \mid j = 0...k\} \tag{4}$$

$$\tau_{\pi,k}(x_i) = \sigma(x_i) \quad \forall x_i \in S(\tau_{\pi,k}) \tag{5}$$

[1] There is some similarity with the idea of pseudo-costs for MIP [1].

i.e., $\tau_{\pi,k}$ forces the first $k+1$ variables in π to assume the value they have in σ.

In other words, our impact measure is simply the variation of the cost lower bound recorded when adding the k-th assignment, during the re-application of the current solution σ in the order specified by π. It is possible to aggregate the cost impacts over a set of dives Π via their average, in this case we have:

$$\mathcal{I}^z(x_i, \sigma, \Pi) = \frac{1}{|\Pi|} \sum_{\pi \in \Pi} \mathcal{I}^z(x_i, \sigma, \pi) \tag{6}$$

A permutation-independent measure could be obtained by aggregating the cost impacts for every possible permutation. Since this would be prohibitive to obtain exactly, we propose to use the average impacts over a finite set of dives as an approximation. How often to perform the dives and how to choose the permutation π for each of them are some of the decisions that must be taken in order to design an actual neighborhood selection heuristic. Specifically, we have experimented with:

- For the diving frequency: 1) n dives per LNS iteration and 2) diving every n LNS iterations (which incurs less overhead).
- For the choice of the permutation: 1) uniformly randomized permutations and 2) decreasing-impact permutations (the variables are sorted in π by decreasing impact in an attempt to spread the cost variations).

Since the cost impacts depend on the incumbent solution, the accumulated impacts must be reinitialized whenever an improving solution is discovered. If no improving solution is found, each dive will add to the aggregated cost impacts, so that they will converge to the real average.

We experimented with different neighborhood selection strategies based on this information. Our most successful approach exploits the impacts for biasing the choice probability of the variables to be *relaxed*. The method is described in Algorithm 3 and consists in drawing a fixed number of variables from X, without replacement. The drawing probabilities are given by a score (see line 1), that in our case is a convex combination of the cost impact and a uniform quantity:

$$s_i = \alpha \cdot \mathcal{I}^z(x_i, \sigma, \Pi) + (1 - \alpha) \cdot \frac{1}{|X|} \sum_{x_j \in X} \mathcal{I}^z(x_j, \sigma, \Pi) \tag{7}$$

The presence of a uniform term ensures that even variables with zero impact have a chance to be relaxed. The strategy has a single additional parameter $\alpha \in [0, 1]$ (with $\alpha = 0$ corresponding to a pure random selection). In our experiments, we use $\alpha = 0.5$. We dive every 10 (failed) LNS attempts and every time an improving solution is found.

4 Experiments

The ability to diversify is one of the main reasons why on some benchmarks a pure random relaxation obtains very good results. For instance, on the Steel

Algorithm 1 Cost Impact Based Probability

```
1: assign a score s_i to each variable (see Equation 7)
2: let r = Σ_{x_i} s_i
3: while not enough variables selected for relaxation do
4:    pick a random value v in [0, r]
5:    for all not selected x_i do
6:       v = v − s_i
7:       if v ≤ 0 then
8:          r = r − s_i
9:          select x_i for relaxation and continue at line 2
```

Mill Slab problem, a pure random relaxation was the best performer in [4, 13]. Mairy et al. also concluded in [6] that their advanced reinforcement based learning strategy does not obtain better results than a random neighborhood on car sequencing problems. Given that our experimentation targets the Steel Mill Slab problem, it was natural to choose a pure random relaxation as a baseline for a comparison. Furthermore, we decided to include in our evaluation an implementation of the Reversed PGLNS from [10], because the strategy was demonstrated to be better than random relaxation on the car sequencing problem.

From the Steel Mill Slabs benchmarks most commonly employed in the literature[2], we selected the instances with 2,3,4 and 5 slab capacities (80 instances in total), because they were found to be the most difficult in [13]. We limited the number of LNS iterations per run to 1,000, so that the best solution was stable enough for each of the 3 considered relaxation strategies. The size of the neighborhood is 5 (i.e. we relax five variables) and each LNS iteration is stopped after 50 backtracks, as in [13]. All experiments were performed using the OscaR solver [9]. We report detailed results on Table 1.

As it can be seen in Table 1, the Cost Impact based relaxation dominates the Random and Reversed PGLNS on most of the instances. Surprisingly PGLNS seems inferior to Random on this problem[3]. For all of our benchmarks, we used the Student's t-test to check the statistical significance of the performance differences. On 75 over 80 instances the Cost Impact based relaxation obtains smaller average costs at a 5% significance level.

The results we obtained with other Cost Impact based relaxation strategies (not reported due to lack of space), confirmed that retaining some diversification ability is a key feature to obtain a good performance on the Steel Mill Slabs Problem. This is achieved by the proposed neighborhood selection method via the inclusion of a uniform term in the variable scores. This set of experiments confirmed also how beating a pure random relaxation strategy is far from trivial on this problem, which stresses the relevance of our results.

[2] The instances and best known results are available at `http://becool.info.ucl.ac.be/steelmillslab`.

[3] However, the description of the PGLNS approach from [10] lacks some details, hence implementation differences may exist.

Table 1. Results obtained on the instances from [13] with 2,3,4 and 5 capacities averages over 100 executions with different seeds.

#capa	instance	0	1	2	3	4	5	6	7	8	9
2	Random	52.98	74.1	177.62	100.02	36.4	86.32	96.87	77.47	**531**	114.34
	PGLNS	55.24	75.76	178	99.64	38.75	97.06	151.72	77.82	**531**	110.1
	Cost Impact	**45.67**	**71.4**	**176.83**	**98.59**	**33.02**	**72.02**	**93.12**	**72.32**	**531**	**107.46**
3	Random	18.05	74.11	30.38	63.6	23	67.58	67.59	78.98	118.84	235.38
	PGLNS	26.01	79.74	36.45	65.64	18.98	75.35	69.67	86.4	133.56	236.59
	Cost Impact	**13.92**	**69.11**	**24.8**	**55.1**	**18.32**	**64.59**	**49.48**	**71.63**	**116.86**	**227.57**
4	Random	38.32	40.94	**33.64**	32.31	16.16	22.1	21.78	26.59	16.03	27.25
	PGLNS	39.36	40.12	41.33	32.2	16.36	22.25	24.05	29.25	18.14	27.5
	Cost Impact	**38.03**	**38.43**	42.59	**28.01**	**13.64**	**16.56**	**14.25**	**20.16**	**11.81**	**23.44**
5	Random	5.93	32.07	15.68	10.19	21.19	18.83	9.09	17.34	14.63	27.43
	PGLNS	7.25	32.51	16.32	13.23	21.25	21.34	12.57	18.58	13.21	28.4
	Cost Impact	**4.51**	**31.38**	**15.28**	**8.64**	**17.99**	**17.72**	**5.96**	**15.52**	**11.54**	**20.95**
#capa	instance	10	11	12	13	14	15	16	17	18	19
2	Random	97.13	118.26	58.03	166.16	**159.63**	**296**	160.08	196.14	65.04	45.09
	PGLNS	112.54	134.58	53.98	199.56	182.39	296.06	194.66	195.73	71.67	45.41
	Cost Impact	**92.69**	**123.66**	**47.2**	**157.81**	172.67	296.06	**159.72**	**195.46**	**60.64**	**45**
3	Random	51.28	50.65	20.93	**84.39**	28.99	47.52	53.99	28.27	63.6	48.9
	PGLNS	49.5	54.52	25.73	84.52	37.49	48.4	55.85	30.89	65.42	56.51
	Cost Impact	**49.01**	**40.26**	**15.53**	85.34	**23.64**	**47.05**	**47.74**	**24.58**	**54.31**	**38.93**
4	Random	27.09	26.46	19.35	42.47	11.45	29.55	43.89	19.62	27.35	14.07
	PGLNS	32.54	33.84	20.99	42.02	19.08	30.89	45.84	19.47	27.53	14.76
	Cost Impact	**25.1**	**22.81**	**9.72**	**36.43**	**11.06**	**25.88**	**36.06**	**9.02**	**20.81**	**13.9**
5	Random	15.97	18.05	35.21	24.97	8.2	26.81	12.12	20.61	31.89	10.21
	PGLNS	17.54	18.72	38.09	28.29	7.92	29.46	12.49	22.03	33.52	11.68
	Cost Impact	**14.02**	**14.61**	**32.54**	**23.16**	**5.68**	**22.07**	**9.21**	**19.77**	**29.94**	**8.51**

5 Conclusion

In this paper, we have introduced the Cost Impact, a measure of the propagation on the cost variable obtained when replaying the incumbent solution in a random or customized order. Obtaining Cost Impacts is easy and requires only basic support from the solver. In particular, our technique still allows to treat the problem constraints as black-boxes. A second contribution, we have described a simple and effective relaxation strategy based on Cost Impacts.

Our contribution can be seen as a mix of the ideas presented in [3], [10] and [7]. As in [7], we rely on the solver propagation to measure the impact on the cost. We also recognize that variables affecting the most the cost should be relaxed similarly to [3]. Finally as for PGLNS [10], our approach is problem independent and does not require to disable the propagation of any constraint when diving.

Our results have illustrated the superiority of the approach on the Steel Mill Slabs problem over a pure random relaxation and an implementation of Reversed PGLNS. Such outcome proves the potential of the proposed technique, providing motivation for future research.

Our method has been explained by representing solutions as assignments of decision variables, but it could easily be extended to more complex branching decisions (such as ordering activities in scheduling). As future work we plan to experiment the method on a broader set of problems, including scheduling variants.

References

1. M Benichou, JM Gauthier, P Girodet, G Hentges, G Ribiere, and O Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1):76–94, 1971.
2. Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *16th European Conference on Artificial Intelligence (ECAI'04).*, pages 146–150, 2004.
3. Tom Carchrae and J. Christopher Beck. Principles for the design of large neighborhood search. In *Journal of Mathematical Modelling and Algorithms*, volume 8, pages 245–270, 2009.
4. Antoine Gargani and Philippe Refalo. An efficient model and strategy for the steel mill slab design problem. In *Principles and Practice of Constraint Programming–CP 2007*, pages 77–89. Springer, 2007.
5. Philippe Laborie and Daniel Godard. Self-adapting large neighborhood search: Application to single-mode scheduling problems. In *Proceedings MISTA-07, Paris*, pages 276–284, 2007.
6. Jean-Baptiste Mairy. Reinforced adaptive large neighborhood search. In *The Seventeenth International Conference on Principles and Practice of Constraint Programming (CP 2011)*, page 55, 2011.
7. Jean-Baptiste Mairy, Pierre Schaus, and Yves Deville. Generic adaptive heuristics for large neighborhood search. *Seventh International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS2010). A Satellite Workshop of CP*, 2010.
8. Laurent Michel and Pascal Van Hentenryck. Activity-based search for black-box constraint programming solvers. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems, CPAIOR 2012*, volume 7298 of *Lecture Notes in Computer Science*, pages 228–243. Springer, 2012.
9. OscaR Team. OscaR: Scala in OR, 2012. Available from `https://bitbucket.org/oscarlib/oscar`.
10. Laurent Perron, Paul Shaw, and Vincent Furnon. Propagation guided large neighborhood search. In Mark Wallace, editor, *Principles and Practice of Constraint Programming - CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 468–481. Springer, 2004.
11. Philippe Refalo. Impact-based search strategies for constraint programming. In Mark Wallace, editor, *Principles and Practice of Constraint Programming - CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 557–571. Springer, 2004.
12. Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, 2006.
13. Pierre Schaus, Pascal Van Hentenryck, Jean-Noël Monette, Carleton Coffrin, Laurent Michel, and Yves Deville. Solving steel mill slab problems with constraint-based techniques: Cp, lns, and cbls. *Constraints*, 16(2):125–147, 2011.
14. Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint ProgrammingCP98*, pages 417–431. Springer, 1998.
15. Alessandro Zanarini and Gilles Pesant. Solution counting algorithms for constraint-centered search heuristics. In *Principles and Practice of Constraint Programming - CP 2007*, volume 4741 of *Lecture Notes in Computer Science*, pages 743–757. Springer, 2007.