Revisiting the Self-Adaptive Large Neighborhood Search

Charles Thomas and Pierre Schaus

Universite catholique de Louvain, ICTEAM institute,
{charles.thomas, pierre.schaus}@uclouvain.be

Abstract. This paper revisits the Self-Adaptive Large Neighborhood Search introduced by Laborie and Godard. We propose a variation in the weight-update mechanism especially useful when the LNS operators available in the portfolio exhibit unequal running times. We also propose some generic relaxations working for a large family of problems in a black-box fashion. We evaluate our method on various problem types demonstrating that our approach converges faster toward a selection of efficient operators.

1 Introduction

Back in 2004, Puget [1] said that CP technology was too complex to use and more research efforts should be devoted to make it accessible to a broader audience. A lot of research effort has been invested to make this vision become true. Efficient black-box complete search methods have been designed [2–8] and techniques such as the embarrassingly parallel search are able to select the best search strategy with almost no overhead [9]. For CP, Puget argued that the model-and-run approach should become the target to reach. The improvements went even beyond that vision since for some applications, the model can be automatically derived from the data [10, 11].

This work aims at automating the CP technology in the context of Large Neighborhood Search (LNS) [12]. This technique consists in iteratively applying a partial relaxation followed by a reconstruction in order to gradually improve the solution to the problem. The relaxation determines constraints to impose to restrict the problem based on the current best solution. Then, the reconstruction (or search) heuristic guides the search in the resulting search space by assigning values to the remaining variables in order to find one or more new solution(s).

Example 1. For example, a *random* relaxation heuristic selects randomly a percentage of the variables to relax and fix the other ones to their assignment in the current best solution. This heuristic can be parametrized by choosing the percentage to relax in a set of values such as $\{10\%, 20\%, 50\%\}$. A *first fail* heuristic with a fixed limit on the number of backtracks can be used as a reconstruction heuristic which can also be parametrized by choosing a limit on the number of backtracks in a set of values such as $\{50\,bkts, 500\,bkts, 5000\,bkts\}$.

The relaxation and reconstruction process continues until some limit in terms of iterations or time is reached. From a local search point of view, CP is thus used as a slave technology for exploring a (large) neighborhood around the current best solution. LNS has been successfully used on various types of problems: bin-packing [13, 14], vehicle-routing [15, 16], scheduling [17–19], etc. Designing good relaxation and reconstruction heuristics with the right parameters is crucial for the efficiency of LNS. Unfortunately this task requires some experience and intuition on the problem to solve.

In order to design an automated LNS, two approaches can be envisioned. A first one would be to recognize the structure of the model in order to select the most suited heuristic from a taxonomy of heuristics described in the literature. This approach, which is used in [20] for scheduling problems, has two disadvantages: 1) some problems are hybrids and thus difficult to classify or recognize, 2) it requires a lot of effort and engineering to develop the problem inspector and to maintain the taxonomy of operators. Therefore, we follow a different approach called Adaptive LNS (ALNS) introduced in [21] which uses a portfolio of heuristics and dynamically learns on the instance which ones are the most suitable. At each iteration, a pair of relaxation and reconstruction heuristics is selected and applied on the current best solution. The challenge is to select the pair having the greatest gradient of the objective function over time (evaluated on the current best solution) based solely on the past executions.

We expand the usage of the Self Adaptive LNS (SA-LNS) framework proposed in [22] on different optimization problems by considering the model as a black-box. Our solver uses a set of generic preconfigured methods (operators) that hypothesize specificities in the problem and leverage them in order to efficiently perform LNS iterations. Given that the operators available in the portfolio are well diversified, we hope to provide a simple to use yet efficient framework able to solve a broad range of discrete optimization problems.

Our contributions to the ALNS framework are: 1) An adaptation of the weight update mechanism able to better cope with unequal running times of the operators. 2) A portfolio of operators easy to integrate and implement in any solver for solving a broad range of problems.

We first explain in Section 2 the principles of the ALNS framework. Then, in Section 3 we present the heuristics implemented as part of our ALNS portfolio. We present the experiments that we conducted and their results in Section 4. Finally, we provide a few concluding remarks and evoke our further research prospects in Section 5.

2 Adaptive Large Neighbourhood Search

Each ALNS operator as well as its possible parameters is associated to a weight. These weights allow to dynamically reward or penalize the operators and their parameters along the iterations to bias the operator selection strategy. Algorithm 1 describes the pseudo-code for an ALNS search. $\Delta c \geq 0$ is the objective improvement and Δt is the time taken by the operator.

Algorithm 1 Adaptive Large Neighborhood Search For a minimization problem

```
s* ← feasible solution
repeat
  relax ← select relaxation operator
  search ← select search operator
  (s', Δt) ← search(relax(s*))
  Δc ← cost(s*) − cost(s')
  weight_relax ← updateWeight(relax)
  weight_search ← updateWeight(search)
  if Δc > 0 then
    s* ← s'
  end if
until stop criterion met
return s*
```

Roulette Wheel selection We use the Roulette Wheel selection mechanism as in [22, 23]. It consists in selecting the operators with probabilities proportional to their weight. The probability $P(i)$ of selecting the i-th operator o_i with a weight w_i among the set of all operators O is $P(i) = \frac{w_i}{\sum_{k=1}^{|O|} w_k}$

Weight evaluation In [22], the authors evaluate the operators ran at each iteration using an efficiency ratio r defined as: $r = \frac{\Delta c}{\Delta t}$. This ratio is then balanced with the previous weight of the operator $w_{o,p}$ using a reaction factor $\alpha \in [0, 1]$: $w_o = (1-\alpha) \cdot w_{o,p} + \alpha \cdot r$. While the reaction factor is important to accommodate the evolving efficiency of the operators during the search, this method does not cope well with operators having different running times. Indeed, operators with a small execution time will evolve faster as they will be evaluated more often. This can lead less efficient operators to be temporally considered better as their weight will decrease slower.

Example 2. Let us consider two operators A and B with running times of respectively 2 and 4 seconds. Both operators start with an efficiency ratio of 10 but after some time in the search, A has an efficiency of $\frac{1}{2}$ and B of $(\frac{1}{4})$. If each operator is separately run for 4 seconds, under a reaction factor of $\alpha = 0.9$; as A will be evaluated twice, its weight will decrease to 0.595 $(0.1 \cdot (0.1 \cdot 10 + 0.9 \cdot \frac{1}{2}) + 0.9 \cdot \frac{1}{2})$. Over the same duration B would be evaluated once and its weight would become 1.225 $(0.1 \cdot 10 + 0.9 \cdot \frac{1}{4})$. While both operators will eventually converge towards their respective efficiency, for a short amount of time, B will have a higher score than A and thus a higher probability to be selected.

This induces a lack of reactivity in the operator selection. In the following, we propose a variation of the weight update rule, more aligned with the expected behavior in case of different runtimes among the operators.

Evaluation window We evaluate the operator based on its performances obtained in a sliding evaluation window $[t^* - w, now]$ where t^* is the time at which the

last best solution was found and w is the window size meta-parameter. The window thus adapts itself in case of stagnation to always include a fixed part of the search before the last solution was found. This ensures that the operator(s) responsible for finding the last solution(s) will not have their score evaluated to 0 after a while in case of stagnation.

For each LNS iteration i, we record the operator used o_i, the time t_i at which it was executed, the difference Δc_i of the objective and the duration of execution Δt_i. We define the local/total efficiency ratio $L(o)/T(o)$ of an operator and the local/total efficiency L/T of all the operators as:

$$L(o) = \frac{\sum_{i|o_i=o \wedge t_i \in [t^*-w, now]} \Delta c_i}{\sum_{i|o_i=o \wedge t_i \in [t^*-w, now]} \Delta t_i} \qquad T(o) = \frac{\sum_{i|o_i=o \wedge t_i \in [0, now]} \Delta c_i}{\sum_{i|o_i=o \wedge t_i \in [0, now]} \Delta t_i} \tag{1}$$

$$L = \frac{\sum_{i|t_i \in [t^*-w, now]} \Delta c_i}{\sum_{i|t_i \in [t^*-w, now]} \Delta t_i} \qquad T = \frac{\sum_{i|t_i \in [0, now]} \Delta c_i}{\sum_{i|t_i \in [0, now]} \Delta t_i} \tag{2}$$

Intuitively, the local efficiency corresponds to estimating the gradient of the objective function with respect to the operator inside the evaluation window. If the operator was not selected during the window, its local efficiency is 0 which might be a pessimistic estimate. Therefore we propose to smooth the estimate by taking into account $T(o)$ normalized by the current context ratio L/T. The evaluation of an operator o is computed as:

$$weight(o) = (1 - \lambda) \cdot L(o) + \lambda \cdot \frac{L}{T} \cdot T(o) \tag{3}$$

with $\lambda \in [0, 1]$ a balance factor between the two terms. As we desire to evaluate the operator mainly based on its local efficiency, we recommend that $\lambda < 0.5$.

3 Operator portfolio

In this section, we present the relaxation and search operators that we propose to be part of the portfolio. All of them operate on a vector of integer decision variables. This list is based on our experience and the features available in the solver used for our experiments. Therefore it should not be considered as exhaustive.

Relaxation Heuristics

- Random: Relaxes randomly k variables by fixing the other ones to their value in the current best solution. This heuristic brings a good diversification and was demonstrated to be good despite its simplicity [24].
- Sequential: Relaxes randomly n sequences of k consecutive variables in the vector of decision variables. This heuristic should be efficient on problems where successive decision variables are related to each other, for instance in Lot sizing problems [25, 26].

- Propagation Guided and Reversed Propagation Guided: Those heuristics are described in [27]. They consist of exploiting the amount of propagation induced when fixing a variable to freeze together sets of variables whose values are strongly dependent on each other.
- Value Guided: This heuristic uses the values assigned to the variables. We have five different variants: 1) Random Groups: Relaxing together groups of variables having the same value. This variant should be efficient on problems where values represent resources shared between variable such as bin-packing problems. 2) Max Groups: This variant relaxes the largest groups of variables having the same values. It can be useful for problems such as BACP or Assembly line balancing [28, 29]. 3) Min Groups: This method relaxes the smallest groups of variables having the same value (which can be single variables). 4) Max Values: It consists in relaxing the k variables having the maximum values. We expect this heuristic to be efficient with problems involving a makespan minimization. 5) Min Values: This heuristic relaxes the k variables having the minimum values. It should be efficient in case of maximization problems.
- K Opt: This heuristic makes the hypothesis that the decision variables form a predecessor/successor model (where variable values indicate the next or previous element in a circuit). It is inspired by the k-opt moves used in local search methods for routing problems. The principle is to relax k edges in the circuit by selecting k variables randomly. The remaining variables have their domain restricted to only their successor and their predecessor in the current best solution in order to allow inversions of the circuit fragments.
- Precedency Based: This relaxation is useful for scheduling problems and hypothesizes that the decision variables corresponds to starting times of activities. It imposes a partial random order schedule as introduced in [17].
- Cost Impact: This operator was described in [24]. The heuristic consists in relaxing the variables that impact most the objective function when fixed.

Search Heuristics A search heuristic explores the search space of the remaining unbounded variables by iteratively selecting a variable and one of its values to branch on. They can be separated into two components: a variable heuristic and a value heuristic. Here are the variable heuristics used:

- FirstFail tries first variables that have the most chances to lead to failures in order to maximize propagation during the search.
- Conflict Ordering proposed in [4] reorders dynamically the variables to select first the ones having led to the most recent conflicts.
- Weighted Degree introduced in [30] associates a weight to each variable. This weight is increased each time a constraint involving that variable fails.

In combination with these variable heuristics, we used different value heuristics which select the minimum/maximum/median/random value in the domain plus the *value sticking* [31] which remembers the last successful assigned values. We also permit a binary split of the domain into $\leq, >$ branching decisions.

4 Experiments

As in [9] we use an oracle baseline to compare with ALNS. Our baseline consists of a standard LNS with for each instance the best combination of operators (the one that reached the best objective value in the allocated time), chosen a posteriori. Notice that this baseline oracle is not the best theoretical strategy since it sticks with the same operator for all the iterations.

We implemented our framework in the OscaR constraint programming solver [32] where it is available in open-source. We tested our framework on 10 different constraint optimization problems with two arbitrarily chosen medium-sized instances per problem. We compare: 1) The original implementation from [22] (denoted *Laborie* here after) with a reaction factor α of 0.9. 2) The variant of [22] proposed in this article (denoted *Eval window*) with a sliding window $w = 10$ seconds and a balance factor λ of 0.05. 3) The oracle baseline.

Each approach was tested from the same initial solution (found for each instance using a first-fail, min-dom heuristic) with the same set of operators. We used relaxation sizes of $\{10\%, 30\%, 70\%\}$ and backtracks limits of $\{50\,bkts$, $500\,bkts$, $5000\,bkts\}$. We generated a different operator for each parameter(s) value(s) combination but kept the relaxation and reconstruction operators separated. We have 30 relaxation and 36 reconstruction operators, which yields a total of 1080 possible combinations to test for the baseline. Each ALNS variant was run 20 times with different random seeds on each instance for 240 seconds. We report our results in terms of cost values of the objective function for each instance. In order to compare the anytime behavior of the approaches, we define the *relative distance* of an approach at a time t as the current distance from the best known objective (BKO) divided by the distance of the initial solution: $(objective(t) - BKO)/(objective(0) - BKO)$. A relative distance of 0 thus indicates that the optimum has been reached.

We report the final results in Table 1. For each instance, we indicate the best known objective (BKO) and the results of the evaluated approaches after 240 seconds of LNS. For each approach, we report the average objective value (obj), the standard deviation (std) if applicable and the relative distance to the best known solution (rdist). The best results between the two evaluated approaches are displayed in bold. Figure 2 plots the average relative distance to the best known solution in function of the search time.

The results seem to indicate (at least on the tested instances) that the weight estimation based on an evaluation window tends to improve the performances of the original ALNS as described in [22]. The average relative distance to the best known solution is of 0.12 at the end of the search using the evaluation window, while it is of 0.18 using our implementation of [22]. None of the ALNS approaches is able to compete with the baseline (except on a few instances), but they obtain reasonably good solutions in a short amount of time. Furthermore, their any-time behavior is good when compared to the baseline and tends to get closer towards the end of the search.

Figure 1 shows a heat map of the relative selection frequency of the relaxation operators for each problem in the Eval window approach. The darker an entry,

Table 1. Experimental results

Instance	Problem	BKO	Baseline		Eval window			Laborie		
			obj	rdist	obj	std	rdist	obj	std	rdist
la13	JobShop	1150.00	1150.00	0.00	**1157.65**	12.06	**0.00**	1195.65	156.4	0.01
la17	(Lawrence-84)	784.00	784.00	0.00	784.05	0.22	0.00	**784.00**	0.00	**0.00**
chr22b	QAP	6194.00	6292.00	0.01	**6517.80**	115.16	**0.04**	6626.60	135.71	0.06
chr25a	(Christofides-89)	3796.00	3874.00	0.00	**4682.00**	393.52	**0.04**	4982.30	342.88	0.06
j120_11_3	RCPSP	188.00	228.00	0.08	420.75	129.61	0.48	**399.55**	62.88	**0.43**
j120_7_10	(Kolisch-95)	111.00	374.00	0.49	160.60	113.23	0.09	**127.70**	1.42	**0.03**
bench_7_1	Steel	1.00	6.00	0.03	**30.65**	11.69	**0.18**	49.75	8.92	0.29
bench_7_4	(CSPLib)	1.00	9.00	0.04	**24.30**	5.83	**0.12**	42.05	8.05	0.21
kroA200	TSP	29368.00	31466.00	0.01	**50022.35**	11159.91	**0.06**	156239.10	12097.24	0.37
kroB150	(Krolak-72)	26130.00	26141.00	0.00	**28596.20**	872.77	**0.01**	61658.70	6051.92	0.14
C103	VRPTW	82811.00	82814.00	0.00	**105876.10**	5553.99	**0.07**	137749.30	17371.97	0.17
R105	(Solomon-87)	137711.00	137261.00	0.00	**140162.30**	1593.46	**0.02**	144273.05	2051.77	0.05
t09-4	Cutstock	73.00	73.00	0.00	**76.65**	3.38	**0.10**	77.10	1.73	0.11
t09-7	(XCSP)	161.00	161.00	0.00	161.00	0.00	0.00	161.00	0.00	0.00
qwhopt-o18-h120-1	Graph colouring	17.00	17.00	0.00	17.00	0.00	0.00	17.00	0.00	0.00
qwhopt-o30-h320-1	(XCSP)	30.00	30.00	0.00	**541.40**	51.90	**0.59**	653.20	32.46	0.72
PSP_100_4	Lot sizing	8999.00	9502.00	0.03	**11796.85**	1204.18	**0.18**	14682.45	847.30	0.36
PSP_150_3	(Houndji-2014)	14457.00	16275.00	0.23	**18236.15**	569.80	**0.48**	19482.20	339.11	0.63
cap101	Warehouse	804126.00	804126.00	0.00	804599.55	428.34	0.00	**804556.50**	430.50	**0.00**
cap131	(XCSP)	910553.00	910553.00	0.00	**913147.60**	2701.98	**0.00**	913240.40	3197.29	0.00
Average				0.05		1246.05	**0.12**		2156.88	0.18

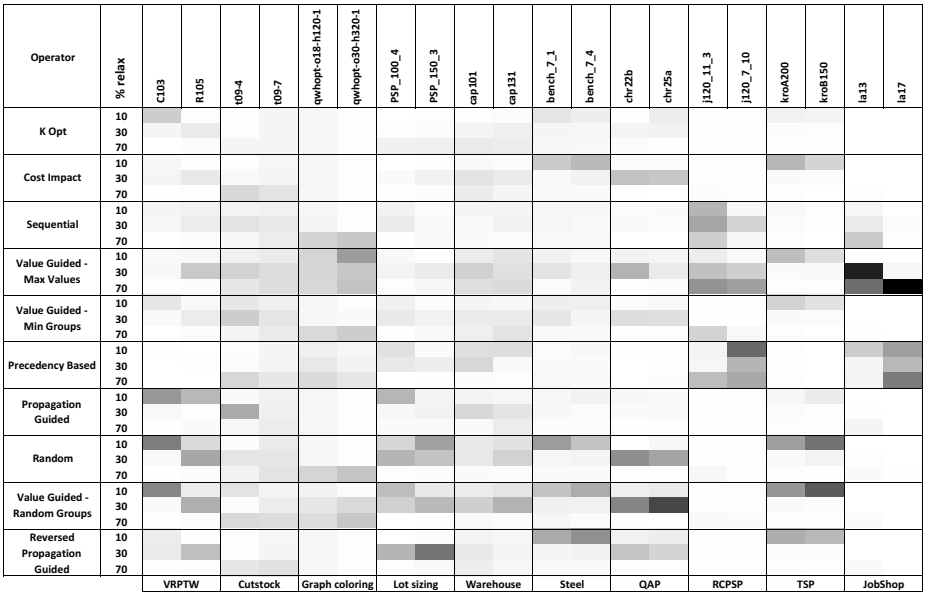

Fig. 1. Heat map of the relaxation operators selection for the Eval window approach

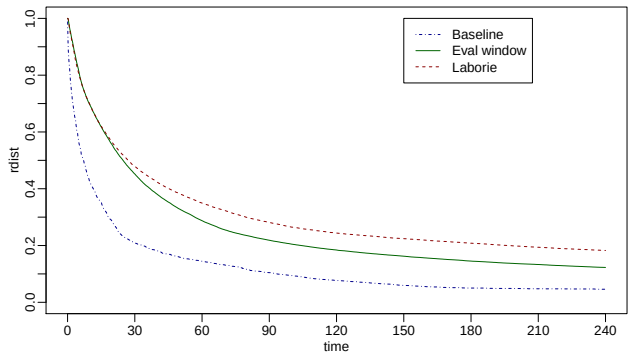

Fig. 2. Average relative distance to BKO during the search

the more frequently this operator was selected for the problem instance. Two interesting observations can be made. First, a subset of operators emerges more frequently for most of the problems. Second, this set varies between problems of different types, but is correlated between instances of the same problem. For some problems this set of operators is more uniform than others. For example, on the warehouse location and the cutting stock problems the operators are selected rather uniformly. The job shop has a strong preference for the max-val and precedency operators. On the contrary, cost-impact is almost useless for the makespan objective of the job shop. Not surprisingly the RCPSP, also a scheduling problem, selects the same two operators as the job shop. The random operator is generally good except for scheduling problems. Due to space limitations, the heat map for Laborie is not given. The selection frequency obtained by the approach of [22] is more uniform, except for scheduling problems for which the same two operators emerge.

The results highlighted by the heat map confirm our intuition and a priori experience, of which operator would be the most successful on each problem. This comforts us that self-adaptive LNS could reach the performances of an expert that would select the operators manually for each problem.

5 Conclusion and future work

The weight update mechanism based on an evaluation window seems a promising adaptation for the original ALNS. In the future we would like to continue researching new relaxation operators for other types of problems (time-tabling, planing, etc) and experiment on a broader set of problems and instances. Parallelizing ALNS would also be an interesting challenge. We believe that ALNS would perform well in solver competitions such as [33, 34] where the set of problems is very broad.

Acknowledgements. We thank the reviewers for their feedback. This work was funded by the Walloon Region (Belgium) as part of the PRESupply project.

References

1. Puget, J.F.: Constraint programming next challenge: Simplicity of use. Principles and Practice of Constraint Programming–CP 2004 (2004) 5–8
2. Refalo, P.: Impact-based search strategies for constraint programming. CP **3258** (2004) 557–571
3. Hebrard, E., Siala, M.: Explanation-based weighted degree. In: International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer (2017) 167–175
4. Gay, S., Hartert, R., Lecoutre, C., Schaus, P.: Conflict ordering search for scheduling problems. In: International conference on principles and practice of constraint programming, Springer (2015) 140–148
5. Chu, G., Stuckey, P.J.: Learning value heuristics for constraint programming. In: International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer (2015) 108–123
6. Michel, L., Van Hentenryck, P.: Activity-based search for black-box constraint programming solvers. Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimzation Problems (2012) 228–243
7. Pesant, G., Quimper, C.G., Zanarini, A.: Counting-based search: Branching heuristics for constraint satisfaction problems. Journal of Artificial Intelligence Research (2012)
8. Vilím, P., Laborie, P., Shaw, P.: Failure-directed search for constraint-based scheduling. In: International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer (2015) 437–453
9. Palmieri, A., Régin, J.C., Schaus, P.: Parallel strategies selection. In: International Conference on Principles and Practice of Constraint Programming, Springer (2016) 388–404
10. Picard-Cantin, É., Bouchard, M., Quimper, C.G., Sweeney, J.: Learning the parameters of global constraints using branch-and-bound. In: International Conference on Principles and Practice of Constraint Programming, Springer (2017) 512–528
11. Beldiceanu, N., Simonis, H.: A model seeker: Extracting global constraint models from positive examples. In: Principles and Practice of Constraint Programming, Springer (2012) 141–157
12. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: International Conference on Principles and Practice of Constraint Programming, Springer (1998) 417–431
13. Malitsky, Y., Mehta, D., OSullivan, B., Simonis, H.: Tuning parameters of large neighborhood search for the machine reassignment problem. In: International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer (2013) 176–192
14. Schaus, P., Van Hentenryck, P., Monette, J.N., Coffrin, C., Michel, L., Deville, Y.: Solving steel mill slab problems with constraint-based techniques: Cp, lns, and cbls. Constraints **16**(2) (2011) 125–147
15. Jain, S., Van Hentenryck, P.: Large neighborhood search for dial-a-ride problems. Principles and Practice of Constraint Programming–CP 2011 (2011) 400–413
16. Bent, R., Van Hentenryck, P.: A two-stage hybrid local search for the vehicle routing problem with time windows. Transportation Science **38**(4) (2004) 515–530
17. Godard, D., Laborie, P., Nuijten, W.: Randomized large neighborhood search for cumulative scheduling. In: In S. Biundo et al (Eds), Proc. International Conference on Automated Planning and Scheduling ICAPS-05, pp81–89, Citeseer (2005)

18. Carchrae, T., Beck, J.C.: Principles for the design of large neighborhood search. Journal of Mathematical Modelling and Algorithms **8**(3) (2009) 245–270
19. Gay, S., Schaus, P., De Smedt, V.: Continuous casting scheduling with constraint programming. In: International conference on principles and practice of constraint programming, Springer (2014) 831–845
20. Monette, J.N., Deville, Y., Van Hentenryck, P.: Aeon: Synthesizing scheduling algorithms from high-level models. In: Operations Research and Cyber-Infrastructure. Springer (2009) 43–59
21. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation science **40**(4) (2006) 455–472
22. Laborie, P., Godard, D.: Self-adapting large neighborhood search: Application to single-mode scheduling problems. Proceedings MISTA-07, Paris **8** (2007)
23. Pisinger, D., Ropke, S.: A general heuristic for vehicle routing problems. Computers & operations research **34**(8) (2007) 2403–2435
24. Lombardi, M., Schaus, P.: Cost impact guided lns. In: International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer (2014) 293–300
25. Fleischmann, B.: The discrete lot-sizing and scheduling problem. European Journal of Operational Research **44**(3) (1990) 337–348
26. Houndji, V.R., Schaus, P., Wolsey, L., Deville, Y.: The stockingcost constraint. In: International conference on principles and practice of constraint programming, Springer (2014) 382–397
27. Perron, L., Shaw, P., Furnon, V.: Propagation guided large neighborhood search. Principles and Practice of Constraint Programming–CP 2004 (2004) 468–481
28. Monette, J.N., Schaus, P., Zampelli, S., Deville, Y., Dupont, P., et al.: A cp approach to the balanced academic curriculum problem. In: Seventh International Workshop on Symmetry and Constraint Satisfaction Problems. Volume 7. (2007)
29. Schaus, P., Deville, Y., et al.: A global constraint for bin-packing with precedences: Application to the assembly line balancing problem. In: AAAI. (2008)
30. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: Proceedings of the 16th European Conference on Artificial Intelligence, IOS Press (2004) 146–150
31. Frost, D., Dechter, R.: In search of the best constraint satisfaction search. (1994)
32. OscaR Team: OscaR: Scala in OR (2012) Available from `https://bitbucket.org/oscarlib/oscar`.
33. Stuckey, P.J., Feydy, T., Schutt, A., Tack, G., Fischer, J.: The minizinc challenge 2008-2013. In: AI Magazine 35. (2014) 55–60
34. Boussemart, F., Lecoutre, C., Piette, C.: Xcsp3: An integrated format for benchmarking combinatorial constrained problems. arXiv preprint arXiv:1611.03398 (2016)