Variable-Relationship Guided LNS for the Car Sequencing Problem

Filipe Souza[1,2](✉), Diarmuid Grimes[2,3], and Barry O'Sullivan[1,2]

[1] Insight SFI Research Centre for Data Analytics, University College Cork, Cork, Ireland
f.desouza@cs.ucc.ie
[2] SFI Centre for Research Training in Artificial Intelligence, Cork, Ireland
[3] Munster Technological University, Cork, Ireland
http://www.ucc.ie/, http://www.crt-ai.cs.ucc.ie, http://www.mtu.ie/

Abstract. Large Neighbourhood Search (LNS) is a powerful technique that applies the "divide and conquer" principle to boost the performance of solvers on large scale Combinatorial Optimization Problems. In this paper we consider one of the main hindrances to the LNS popularity, namely the requirement of an expert to define a problem specific neighborhood. We present an approach that learns from problem structure and search performance in order to generate neighbourhoods that can match the performance of domain specific heuristics developed by an expert. Furthermore, we present a new objective function for the optimzation version of the Car Sequencing Problem, that better distinguishes solution quality.

Empirical results on public instances demonstrate the effectiveness of our approach against both a domain specific heuristic and state-of-the-art generic approaches.

Keywords: LNS · Neighbourhood selection · Car sequencing problem

1 Introduction

Large Neighbourhood Search (LNS) [14] is a powerful technique to tackle Combinatorial Optimisations Problems, but its main drawback remains on the necessity of an expert to refine the algorithm components for the specific behaviour of each problem. One of the most crucial components is the neighbourhood selection approach, which is highly sensitive to the characteristics of the given problem. Thus, an important open research question concerns developing generic neighborhood selection heuristics that can be efficient in a broad range of problems. Even though it is hard to imagine that a generic approach can overcome a domain specific heuristic designed accurately by an expert, generic approaches

Supported by SFI Centre for Research Training in Artificial Intelligence under Grant No. 18/CRT/6223 and SFI under Grant No. 12/RC/2289-P2, co-funded under the European Regional Development Fund.

have an essential role to popularize LNS as one of the most powerful technique to solve a broadly range of complex large-scale Combinatorial Optimisations Problems (COP).

Freuder and O'Sullivan [3] proposed a number of grand challenges for Constraint Programming (CP). A common aspect amongst these challenges is the requirement for approaches that can solve a range of different problems without the need of a human expert to fine tune its parameters or design dedicated algorithms. Following this line, in this paper we present a novel constraint-based Large Neighbourhood Search that learns from problem structure and search performance in order to create complex and diverse neighbourhoods without the need of a domain specific algorithm. We hypothesise that good neighborhoods can be identified through combining information regarding the problem structure with information learnt during search.

2 Related Work

Large neighborhood search, Fig. 1, was first proposed by Shaw in 1998 [14] as a means of applying CP techniques to large vehicle routing problems. In its basic form, an initial solution is generated and then refined in successive iterations. Each iteration involves firstly the selection of a subset of variables (the neighborhood), whose assignment is relaxed while all other variables have their assignment fixed to the value in the current solution. The neighborhood of unassigned variables can then be solved using a systematic approach, like CP or MIP, to find the optimal solution to the neighborhood given the assignment of the non-neighborhood variables. A key aspect, as highlighted in the figure, is how the neighborhood is selected in each iteration.

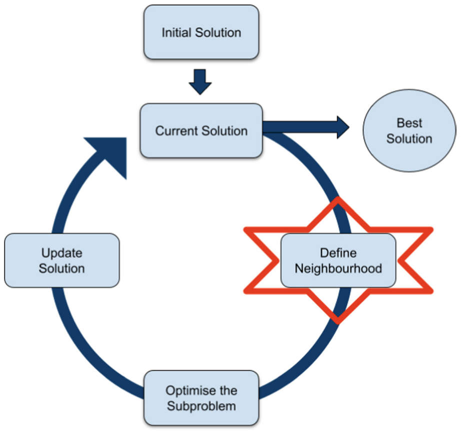


Fig. 1. Large neighbourhood search

2.1 LNS: Domain Independent Neighbourhood Heuristic

A large number of dedicated neighborhood operators have been proposed for different problems. Our focus in this work concerns approaches which have tried to create more generic LNS neighborhood selection methods. To date, much of the literature in this area has focused on portfolio approaches to automatically define the best neighbourhood selection heuristic from a predefined list. For example Laborie and Godard [6] proposed to tackle 21 variations of Single-Mode Scheduling Problems by applying a reinforcement learning method to select, in each iteration, the most suitable neighbourhood selection heuristic for a given instance from a predefined portfolio. The main drawback of this approach can be observed when heuristics from the portfolio have different run-times. Under these circumstances the heuristics with smaller run-time will be used more often as the reward function is given by $\Delta c/\Delta t$ i.e. the size of improvement in objective value divided by the run-time to achieve that improvement. Typically large improvements can be made earlier on while as we approach the optimal solution the improvements are smaller.

In order to address this drawback, Thomas and Schaus [16] proposed a new weight-update mechanism for the portfolio approach. This mechanism works by evaluating the neighborhood heuristic based on its performances obtained in an evaluation window which starts on β iterations before last improvement until the current iteration. That way the windows will always keep information from a fixed part of the search before any stagnation. Even though this approach proved its efficiency on a broad range of problems, we believe that there are two points of further investigation. Firstly, in the results presented in [16], we observe that the random neighbourhood selection performs well in a wide range of problems, even when compared to more sophisticated approaches (we will elaborate on this in the following section). Secondly, these approaches are highly dependent on the list of neighbourhood selection heuristics in the portfolio, thus they cannot be fully classified as domain independent approaches.

To the best of our knowledge the first effective domain independent approach was the Propagation Guided Large Neighbourhood Search (PGLNS) [11]. Here Perron et al. proposed choosing the neighborhood variables based on analysis of the impact of each frozen variable in turn. They tested a number of configurations, however the approach that performed better than the domain specific neighbourhood selection heuristic Interval-Based [10] was a configuration that alternated between three neighborhood selection methods. In one, neighbourhoods are created by starting from an empty solution and incrementally freezing variables based on the propagation impact until achieving the desire neighbourhood size. The second neighbourhood method built by starting from a complete solution and incrementally relaxing variables based on the propagation impact until achieving the desire neighbourhood size. The final neighborhood was generated randomly. The first two approaches are highly efficient to learn from variable relationships, but they do not use information about the variables behaviours during the search process, which we believe to be highly beneficial to generate better neighbourhoods. For instance, when a variable is already assigned to its optimal value, or already has been selected many times, there is no reason to keep selecting this variable only because it has a strong relationship with other variables.

On the other hand there are some domain independent approaches that focus only on variables behaviours aspects, and do not consider the structural relationship between variables. Carchrae and Beck in [2] proposed a Cost-based method to select neighbourhoods, where the variable impact on the overall objective function is the main component to select the variables that will compose the neighbourhood. Their results demonstrated the importance of a stochastic element to ensure a high variety of neighbourhoods, mainly when the instance problem is not so large. Lombardi and Schaus [9] also proposed a heuristic that relies on the cost impact capability of the variables. Their calculation of cost impact is based on lower bound cost before and after assigning the variable its value from the current best solution during a permutation of orderings of the variables. These impacts are then used to weight a roulette wheel style selections strategy for the neighborhood operator.

2.2 Exploration vs Exploitation on Neighbourhood Selection

Many LNS approaches in the literature have reported impressive performance of the simple random neighbourhood selection method, even when we compare it with more sophisticated heuristics, [2,11,12,16]. A highly deterministic approach may choose very similar neighborhoods multiples times resulting in a huge computational time spent on neighbourhoods that do not have as much capacity for improvement; while ignoring some parts of the search space. On the other hand, a complete stochastic approach has a poor exploration of any knowledge from the problem, variables and their connections, but if the neighbourhoods are relatively large, the likelihood of selecting a small number of connected variables where lies some improvement is considerable.

3 Problem Definition

The Car Sequencing problem was originally defined as a Constraint Satisfaction Problem (CSP) [1,15] that aims to allocate a set of cars on a production line of options' installation over a fixed number of timeslots (e.g. one day of timeslots). Each bay has its own capacity, i.e. the number of cars they can work on in a segment of the production line. Furthermore each bay can install only one type of option.

In order to transform this problem to a Constraint Optimisation Problem (COP), we add a new class of car where no option is needed, similar to [10,11]. They used the concept of empty slots providing buffers which are then to be minimised, i.e. minimise the number of extra time slots needed to allocate all cars. The novelty in our formulation is the use of the number of options not placed on the original production line as objective function, with the logic that cars with fewer option requirements would be easier to slot in on a subsequent day.

This approach allows the search to distinguish between two partial solutions even when both have the same number of original cars placed on the production line by prioritising the solution where the placed cars have more options installed, since the cars with less options installed are more likely to find a place in the following iterations.

The problem can be more formally defined as follows:

Definition 1 (Option). *An option $o \in O$ is an extra item to be installed on some specific configuration of a given car, e.g. Parking Assist, Speed Limit Assist, Air Conditioning.*
Option o is characterized by: the window size, WS_o, on the production line; and the maximum number of the option, MC_o, that can be installed in the window.

Definition 2 (Configuration). *A configuration $c \in C$ is a version of a car with a particular set of options. c is characterized by: the number of needed cars $cars_c$; and $REQ_{c,o} \forall o \in O$ that defines whether an option is required by the configuration.*

Definition 3 (Position). *A position $p \in P$ is a place in the queue of the car production line.*

Definition 4 (Solution). *A solution S is an assignment of $\forall c \in C$ to a position $p \in P$. We will formally represent the assignment by $PC_p = c$.*

$$\text{minimize } \left(\sum_{c \in C} \sum_{o \in O} (REQ_{c,o}) * cars_c\right) - \sum_{p \in P} \sum_{o \in O} (REQ_{PC_p,o}) \tag{1}$$

subject to:

$$\sum_{p \in P} (PC_p = c) <= cars_c \qquad \forall c \in C \tag{2}$$

$$\sum_{j=p}^{p+WS_o} REQ_{PC_j,o} \leq MC_o \qquad \forall p \in P, \forall o \in O \tag{3}$$

Constraint 2 guarantees that for each configuration c the maximum number of produced car is $cars_c$. While constraint 3 ensures that no bay is overloaded. In other words, for an option o on any sequence of WS_o cars, the maximum number of these cars that requires option o is MC_o.

4 Neighbourhood Selection Heuristic

The method of neighbourhood selection is a key component in any LNS technique. Using an efficient heuristic to select the next set of neighbours that have high probability of being optimised can greatly increase performance. However, a deterministic approach can result in ignoring some parts of the search space and end up with a relatively poor local minimum solution. On the other hand, a completely stochastic approach may spend a huge computational time on neighbourhoods that do not have scope for improvement.

Our proposed approach exploits the structural relationship between decision variables to guide the search process towards connected neighbourhoods, and information learnt during search to try to choose neighbourhoods with high likelihood of improvement.

Algorithm 1: Neighbourhood Selection Heuristic

```
$randomVars \leftarrow selectNRandomVars(NRandomVars)$;
$bestVar \leftarrow selectBestVar(randomVars)$;
$relaxVar(bestVar)$;
while $checkSize()$ do
    $randomVars \leftarrow selectNVarsRelatedTo(NRandomVars, bestVar)$;
    $bestVar \leftarrow selectBestVar(randomVars)$;
    $relaxVar(bestVar)$;
end
```

Algorithm 1 describes our domain independent neighbourhood selection heuristic. The heuristic works by incrementally relaxing variables selected according to one of the criteria described in the next subsection. In order to maintain a greater degree of diversification, we first select a random subset of variables (*selectNRandomVars*), and then choose the best amongst this according to the criteria (*selectBestVar*). After a variable is selected, the next selection is constrained to the variables that are involved in constraints with the variables already selected (*SelectNVarsRelatedTo*), except for global constraints that involve more than half of the decision variables. It should be noted that the benefit of the selectNRandomVars function isn't just an increase in diversification, it also reduces the computational effort as we only need to find the best amongst this subset as opposed to the best amongst all variables.

4.1 Neighborhood Heuristics

We investigated the performance of the following four heuristic criteria:

Weighted Variable Usage (*V_Usage*): This heuristic prioritises diversification. Each variable has a counter which is incremented when the variable is chosen in the neighborhood of an iteration. The heuristic biases selection to those that have been chosen the least. However, in order to not penalise centroid variables that have to change their values in order to allow other variables to be able to change to the optimal value, the usage score is divided by the number of times that the variable changes its value after a sub-problem optimisation that improves the whole solution. Therefore the criteria is choose the variable with minimum value of *usage/improvements*.

Weighted Variable Cost (*V_Cost*): The criteria for this heuristic is the impact of a variable on the cost (objective function). This cost is calculated by measuring the impact of removing each variable from the current solution.

This heuristic considers the hypothesis that the best neighborhood should involve variables with the higher cost associated with them. The fundamental difference between our variable cost score and the variable cost score from Lombardi and Schaus [9] is that while Lombardi's approach is calculated based on variation of lower bound cost during a range of re-application of the current best solution on a sample of permutation ordering of the variables, our variable cost is calculated based on the impact of unassigning the variable from a full solution.

Weighted Variable Conflicts (*V_Conflicts*): The number of conflicts that each variable was involved on previous iterations. The hypothesis on using this weight is that variables involved in many constraint conflicts are the ones most difficult to find their optimal values, so if we find the optimal values for these variables the others will be easily optimised. We use the variable conflict score implemented on Gecode solver, that is calculated based on the definitions of Conflict history base for SAT problem [7,8]. For more detail of how the variable conflict is calculated in Gecode, please see [5].

Weighted Variable Failures (*V_Fails*): The number of leaf failures that each variable was involved on previous iterations divided by the variable domain size after being relaxed and propagating arc-consistency based on the fixed variables on a given iteration. We hypothesise that variables with a high number of failures are the ones most difficult to find their optimal values, so they need to be relaxed more frequently. For more details of how the Failures criteria is calculated in Gecode, please see [13].

For the latter three approaches, the scores were normalized by dividing it by the number of times that the variable was relaxed in the previous iterations. Therefore, impact of behavior in earlier iterations will not dominate.

5 Evaluation

5.1 Experimental Design

We implemented our proposed approach using Gecode 6.2 [4]. For comparison, we have also tested 4 neighbourhood selection heuristics from the literature: PGLNS[1] [11], Interval-based [10], Cost-Impact (referred to as *CGLNS* in results) [9], and Random which simply chooses the variables for the neighborhood randomly.

It should be noted that we are using the best configuration presented in [11], that iterates through the following three neighborhood operators: Propagation Guided; Reverse Propagation Guided; and purely random selection. However, we defined the neighbourhood size based on the number of relaxed variable

[1] The description of the PGLNS approach from [11] miss some details, thus implementation differences may exist.

Table 1. Configurations parameters for the benchmark experiment.

Parameter	Value
Runtime	120 s
Neighbourhood size	10 slots
Failure threshold	200
NRandomVars	10

instead of the search space size, in order to compare all approach on the same neighbourhood size.

The experiments were run on a Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-70-generic) with 16 Core and 32 GB, with a runtime cutoff of 2 min per approach on each instance. Furthermore, as all approaches have a strong stochastic element, the presented results are the average of 5 runs with different seeds. Table 1 presents the parameter configurations that was used to run the experiments.

5.2 Instances

The experiments will use the three sets of hard instances available on the CSPLib [15]. There are 10 instances in each set and the total number of cars per instance is 200, 300 and 400 respectively. This allows us to also empirically analyse the scalability performance and behaviour of each approach as problems grow in size.

5.3 Results

The results are presented in Table 2 in terms of average cost across the five runs, and the associated standard deviation of the cost. We further provide details on the number of best solutions found for each approach per problem set.

We see that all approaches provide significant improvement over the initial solution. Amongst the comparison approaches, PGLNS performs best with significant improvements over the two other domain-independent heuristics (Random and CGLNS), and was consistently better than the problem-specific heuristic (Interval) on all problem sizes.

Amongst the approaches proposed in this work, we find that all perform well. Although PGLNS outperforms them on the smallest instance size, as the instance size increases, so does the improvement over PGLNS for each. This can be seen more clearly in the number of total best solutions found. For the smallest instances, PGLNS finds most, but can only find one for the largest instances compared to six for V_Usage.

The *V_Fails* heuristic achieved the best results on the two largest set of instances (300 and 400 cars), while on the set of instance with 200 cars, PGLNS and Interval Based found better solutions on average. Interestingly we did not find a large difference in behaviour of our four heuristics in terms of solution

Table 2. Five runs on three problem sets of 10 instances with 120 s cutoff per instance run. Results per problem set in terms of: average and standard deviation of cost; and number of instances for which a method found best solution across methods tested.

	Average cost			Standard deviation cost			Total best solution		
Approach	Size$_{200}$	Size$_{300}$	Size$_{400}$	Size$_{200}$	Size$_{300}$	Size$_{400}$	Size$_{200}$	Size$_{300}$	Size$_{400}$
Initial Sol	92.0	142.7	187.6	6.9	13.1	11.0	–	–	–
Random	19.6	33.9	45.2	4.6	8.5	6.8	0	0	0
Interval[10]	15.6	27.7	34.6	4.1	6.0	6.2	4	2	0
PGLNS[11]	**15.4**	27.1	33.9	3.9	6.3	5.9	4	3	1
CGLNS[9]	38.4	64.8	72.3	30.7	52.8	55.7	1	0	0
V_Conflict	16.5	26.6	**33.0**	3.6	6.3	6.7	2	4	3
V_Fails	16.6	**26.5**	**33.0**	3.5	6.3	6.6	3	3	2
V_Cost	15.9	26.6	33.3	3.6	6.0	5.4	3	6	3
V_Usage	16.5	26.7	33.1	3.5	6.2	6.7	3	3	6

quality. This suggests the importance of the concept of "variable relationship" which underpins all our heuristics.

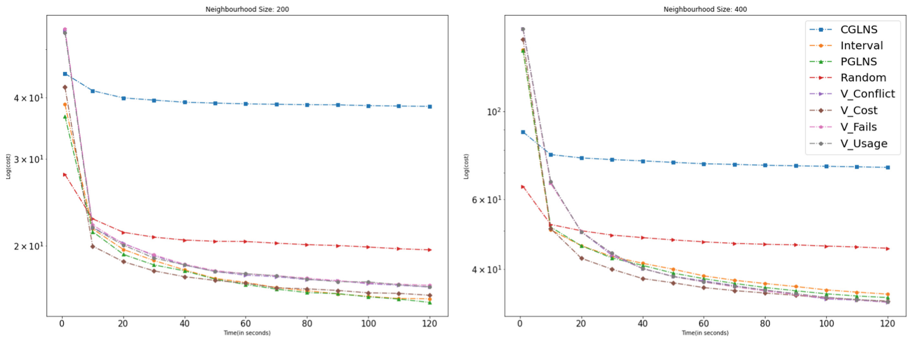
We also generated search statistics, Table 3, for the different approaches in order to gain further insight and understanding into their behavior. These results answer a number of questions. Firstly, we see that both Random and CGLNS performed significantly more iterations than the other approaches, between 5 and 10 times as many. This explains the surprising result that CGLNS had significantly worse performance than even the random neighborhood selector, despite exploring approximately four times as many iterations as PGLNS or Interval.

Table 3. Analysis of search behaviour for different methods averaged across runs and instances.

	Iterations			Nodes per iteration		
Approach	Size$_{200}$	Size$_{300}$	Size$_{400}$	Size$_{200}$	Size$_{300}$	Size$_{400}$
Random	80, 422	53, 958	33, 155	13	8	11
Interval	12, 493	8, 687	5, 849	207	203	204
PGLNS	12, 086	9, 662	6, 051	195	193	199
CGLNS	57, 967	38, 818	25, 227	27	24	25
V_Conflict	6, 664	4, 898	3581	376	374	372
V_Cost	9, 185	5, 965	3782	298	312	328
V_Fails	7, 285	4, 516	3589	376	374	373
V_Usage	6, 501	4, 511	3897	376	374	374

This may seem counter-intuitive, more iterations means more neighborhoods explored which sounds in theory like it should be beneficial. The reason for this not being the case for Random and CGLNS is that many of the neighborhoods selected had scope for very few, if any, variable changes. This is evidenced by the average nodes explored per iteration by these two approaches in comparison to the other approaches. Neither of these approaches take into account the relationship between variables selected. In other words, they relax disconnected variables, and propagating the assignment of the non-relaxed variables results in the domains of most relaxed variables reducing to the previous value.

On the other hand, we see the opposite is the case for the approach we propose, irrespective of the heuristic criteria. Comparing to PGLNS and Interval, our approaches performed over 25% fewer iterations, but explored nearly twice as many nodes per iteration on average. All our approaches use the concept of "variable relationship" in order to build out a connected subset of variables from the initial variable selection. We see a consistent trend whereby this resulted in exploring more nodes per iteration, as more combinations could be tried since the relaxed variables were connected and were not having their domains as restricted by the non-frozen variables.



(a) 200 cars (b) 400 cars

Fig. 2. Evolution of solution quality across time for different instance sizes.

Figure 2 shows the evolution of average cost improvement over 10 instances across 2 min of search for the different configurations described above. We note that CGLNS and more so Random were able to make larger improvements in the first second as they explore more neighborhoods and improvements over the weak initial solution are easy to achieve. However, they quickly stagnate.

As we can see, and showed already in Table 2 PGLNS and Interval-Based have better performance overall on the instances of smallest size (200 cars), whilst our Variable-Relationship neighbourhood selection heuristics got better results on the instances of largest size (400 cars) over 120 s. This behaviour was somewhat expected, since on small size instances it is not essential to prioritise variables with high probability of generated big improvement, as the 120 s search time can guarantee enough iterations to investigate many of the possible neighbourhoods. However, as the instance size increases, the need to prioritise more promising neighbourhoods in each step of the search process also increases, and the Variable Relationship heuristics work better.

We can observe this behaviour in the first 40 s of Graph 2a where the *V_Cost* heuristic got the best performance as it prioritise neighbourhoods with variables that have more impact on the objective function. Interestingly *V_Cost* was significantly better than other approaches after 40 s for both instance sizes, and indeed for size 300 (not shown). This may in part be the combination of searching more neighborhoods than our other heuristics (as evidenced by greater number of iterations in Table 3) while still keeping the Variable-Relationship. These results suggest alternating between our different neighborhood operators may produce better results.

6 Conclusion and Future Works

In this paper, we proposed an approach that combines knowledge extracted from the problem structure and search state information to generate complex and diversified neighbourhoods without the need of a domain specific algorithm. Our heuristic works by incrementally relaxing variables based on its state, and their relationship to other variables selected. In particular, after each variable is selected by the heuristic, the next selection is constrained to the variables that are involved in constraints with the variables already selected.

We empirically evaluated our approach using public instances of Car Sequencing Problem [15]. Comparing our results against domain specific heuristic, SOA generic approaches, and pure random relaxation demonstrated the effectiveness of Variable-Relationship Guided LNS mainly on large instances. Further analysis of search behaviour, in terms of average nodes explored per iteration, provided insight into why these approaches performed so well.

To the best of our knowledge the Variable-Relationship Guided LNS is the first domain independent neighbourhood selection heuristic to combine information from problem structure and that learnt through search performance. Even though the empirical results prove that good neighborhoods can be identified through combining information regarding the problem structure with information collected during search on an optimisation version of the Car Sequencing Problem, there are some promising avenues for future work such as:

- The combination of different types of search state information in the same search process. The main challenge here is to define the relative importance of each information in the construction of the neighbourhood. We believe that Machine Learning/Deep Learning are key to address this challenge in a generic and adaptive way.
- Identifying more complex variable relationships (i.e. for variables not directly connected by a constraint). Graph Convolution Networks could be beneficial to learn more robust relationships of variables based on the graph representation of the constraint relationship between decision variables.

References

1. Artigues, C., Hebrard, E., Mayer-Eichberger, V., Siala, M., Walsh, T.: SAT and hybrid models of the car sequencing problem. In: Simonis, H. (ed.) CPAIOR 2014. LNCS, vol. 8451, pp. 268–283. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07046-9_19
2. Carchrae, T., Beck, J.C.: Cost-based large neighborhood search. In: Workshop on the Combination of Metaheuristic and Local Search with Constraint Programming Techniques (2005)
3. Freuder, E.C., O'Sullivan, B.: Grand challenges for constraint programming. In: Constraints, pp. 1–13 (2014). https://doi.org/10.1007/s10601-013-9155-1
4. Gecode Team: Gecode: Generic constraint development environment (2006). http://www.gecode.org
5. Habet, D., Terrioux, C.: Conflict history based heuristic for constraint satisfaction problem solving. J. Heuristics **27**(6), 951–990 (2021). https://doi.org/10.1007/s10732-021-09475-z
6. Laborie, P., Godard, D.: Self-adapting large neighborhood search: application to single-mode scheduling problems. In: Proceedings MISTA-07, Paris 8 (2007)
7. Liang, J., Ganesh, V., Poupart, P., Czarnecki, K.: Exponential recency weighted average branching heuristic for sat solvers. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30, no. 1 (2016)
8. Liang, J.H., Ganesh, V., Poupart, P., Czarnecki, K.: Learning rate based branching heuristic for SAT solvers. In: Creignou, N., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 123–140. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40970-2_9
9. Lombardi, M., Schaus, P.: Cost impact guided LNS. In: Simonis, H. (ed.) CPAIOR 2014. LNCS, vol. 8451, pp. 293–300. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07046-9_21
10. Perron, L., Shaw, P.: Combining forces to solve the car sequencing problem. In: Régin, J.-C., Rueher, M. (eds.) CPAIOR 2004. LNCS, vol. 3011, pp. 225–239. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24664-0_16
11. Perron, L., Shaw, P., Furnon, V.: Propagation guided large neighborhood search. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 468–481. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30201-8_35
12. Pisinger, D., Ropke, S.: Large neighborhood search. In: Handbook of Metaheuristics, pp. 399–419. Springer, Heidelberg (2010)
13. Schulte, C., Tack, G., Lagerkvist, M.Z.: Modeling and programming with gecode. Schulte, Christian and Tack, Guido and Lagerkvist, Mikael, vol. 1 (2010)
14. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M., Puget, J.-F. (eds.) CP 1998. LNCS, vol. 1520, pp. 417–431. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-49481-2_30
15. Smith, B.: CSPLib problem 001: Car sequencing. http://www.csplib.org/Problems/prob001
16. Thomas, C., Schaus, P.: Revisiting the self-adaptive large neighborhood search. In: van Hoeve, W.-J. (ed.) CPAIOR 2018. LNCS, vol. 10848, pp. 557–566. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93031-2_40