Reinforced Adaptive Large Neighborhood Search

Jean-Baptiste Mairy[1], Yves Deville[1], and Pascal Van Hentenryck[2]

[1] ICTEAM institute, Université catholique de Louvain, Belgium
[2] Brown University, USA

Abstract. The Large Neighborhood Search metaheuristic for solving Constrained Optimization Problems has been proved to be effective on a wide range of problems. This Local Search heuristic has the particularity of using a complete search (such as Constraint Programming) to explore the large neighborhoods obtained by relaxing a fragment of the variables of the current solution. Large Neighborhood Search has three parameters that must be specified (size of the fragment, search limit and fragment selection procedure). Its performances greatly depend on those parameters. Despite the success of the metaheuristic, no generic principle has emerged yet on how to choose the parameters. They are currently set either with domain dependent heuristics or chosen randomly. The objective of this ongoing work is to develop generic heuristics for adaptive selection of the parameters of Large Neighborhood Search. This paper proposes to use a Reinforcement Learning framework in order to adapt the heuristics during the search. Two heuristics are proposed to deal with the first two parameters of the metaheuristic. Three are proposed to adapt the last parameter. Preliminary computational results on the Car Sequencing problem are given. On this problem, only the adaptive selection of the first two parameters is effective.

1 Introduction

One of the major challenges in computer science is to find optimal solutions to hard combinatorial problems. Exact approaches have the advantage of finding solutions that are proved to be optimal. However, such techniques are often too slow to be used on large instances. Local Search techniques are designed to find quickly good solutions. Those are not optimal or not known to be. The Large Neighborhood Search (LNS) metaheuristic is one such Local Search technique. It has been proved to be effective on numerous problems (see for instance [1–5, 8]). Despite being a Local Search technique, the LNS approach uses a complete technique, such as Constraint Programming, to explore its neighborhoods. It takes advantage of the exhaustivity and expressiveness of Constraint Programming and the speed of Local Search.

The LNS metaheuristic has three main parameters that have to be fixed when designing an LNS model. The efficiency of the metaheuristic largely depends on those choices. The parameters can be chosen specifically for the problem, or a simple random strategy can be used. The specific approach has the advantage of being quite effective. However, it requires some knowledge of the problem. Even

for well-known problems, setting the right values for the parameters is a difficult task. Moreover, those strategies are only applicable to the problem they were designed for. Random parameters have the advantage of being totally generic but can be significantly less performant.

The objective of this ongoing research is to develop generic heuristics for an adaptive selection of the LNS parameters. This work uses Reinforcement Learning to allow the heuristics to be adapted during the search.

The remaining of this paper describes different applications of Reinforcement Learning to LNS. Section 2 presents the Large Neighborhood Search metaheuristic. Section 3 describes the Reinforcement Learning framework used through this work. Section 4 presents the techniques developed to deal with the first two parameters of LNS. Section 5 then presents the application of reinforcement learning to the remaining parameter of LNS. Preliminary results on the car sequencing problem are given in Sections 4 and 5. On this problem, the adaptive selection of parameters is only effective on some of these parameters. Section 6 compares the heuristics to the results of the state of the art generic LNS heuristic. Some perspectives of this ongoing research are described in Section 7.

2 Large Neighborhood Search

Large Neighborhood Search (LNS) combines Local Search and Constraint Programming. Local Search working principle is to always maintain a candidate solution. This current solution is not optimal or not known to be or even violates the constraints. At each iteration, a neighborhood of the solution is searched to hopefully find an improving solution. If such a solution is found, it becomes the current solution. The neighborhoods are defined by perturbing the current solution. In Large Neighborhood Search, the current solution is not violating any hard constraint. The neighborhoods are defined by relaxing a subset (called fragment) of the variables to their original domain. The rest of the variables are fixed to their value in the current solution. An improving solution that does not violate the hard constraints is searched in the defined neighborhood using Constraint Programming. LNS thus consists of the repetition of the following two steps until a stopping criterion is met.

1. *neighborhood definition*: choosing the fragment of variables that will be relaxed to their original domains.
2. *neighborhood exploration*: using CP to explore the restricted problem defined by the relaxation of the fragment. A limit in number of failures is specified to avoid spending too much time in exploring the neighborhood.

LNS requires three parameters: (1) the size of the fragment, (2) the fragments selection procedure and (3) the limit on the CP exploration step. Although the LNS framework is really simple, its parameters must be carefully chosen. They can evolve through time. No general principle has emerged yet on how to choose them from one restart to the other. Practitioners often prefer a fixed size of the fragments, a fixed search limit and a random selection of the variables to relax. The random selection favors diversification.

3 Reinforcement Learning framework

This section presents the Reinforcement Learning framework that is used in this work to generically deal with the LNS parameters.

Reinforcement Learning [6] can, in a nutshell, be summarized as choosing actions and observing the results. In the LNS metaheuristic, the actions are the selection of a fragment of a given size and a CP search limitation.

The general picture of Reinforcement Learning uses the notion of *agent*. The task of the *agent* is to select *actions*. For each *action* chosen, it gets a *reward*. Its goal is to maximize those *rewards*. The best *action* to choose varies given the *state* of the *external world*. Each *action* results in a *reward* but also in a new *state* for the *external world*. In order to choose the *actions*, the agent maintains *internal quantities*. Those quantities usually reflect the quality of the states and state-action pairs. They are updated each time a new *reward* is obtained. The name Reinforcement Learning comes from the fact that those quantities are used to select *actions* while being updated with respect to the result of the selected *actions*. All the methods described in this paper use the view of Reinforcement Learning depicted in Fig. 1. The parameter selector selects parameters based on the internal quantities that it maintains. It updates those quantities based on the rewards. Here, there is only one state for the external world. The rewards are defined with respect to the output of the CP search.

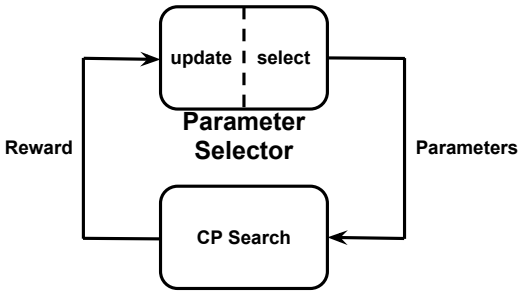

Fig. 1. Reinforcement Learning framework for LNS

The next sections present the application of this framework to the selection of the LNS parameters. All the techniques are described in terms of their internal quantities, the computation of the rewards, the update of the internal quantities and the selection of the parameters. The rewards are computed with respect to the improvement of the objective value.

4 Fragment Size and Search Limit Selectors

This section describes two different fragment size and search limit selectors. As those two parameters are dependent on each other, the selectors deal with both of them. This section starts by describing an uninformed parameter selector. Then it describes an informed selector. Some preliminary computational results are given at the end of the section.

4.1 Uninformed Selection of Parameters

This section presents the first heuristic developed to deal with the choice of the fragment size and the CP search limit. This heuristic is called uninformed because it does not use any knowledge about the parameters. It is similar to the n-armed bandit method described in [6].

The two parameters are not independent. With a large fragment size, the search space is intended to be large. The search limit should then be set accordingly. For smaller fragment size, the search limit is not as important as the search space is small. However, some improvements on the objective require large fragment sizes.

Internal Quantities: for each fragment size fs and search limit sl, the heuristic maintains:

$$Q[fs, sl] : \text{the quality of the pair } (fs, sl) \tag{1}$$

All those quantities are positive during the entire search. The possible fragment sizes and search limits lie in ranges given as parameters to the technique.

Rewards: the reward system is

$$reward = \begin{cases} 50 & \text{if objective improved} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Updates: the updates use exponential smoothing for the previously used pair of fragment size and search limit (fs, sl):

$$Q[fs, sl] := Q[fs, sl] + \alpha * (reward - Q[fs, sl]) \tag{3}$$

where α is a parameter of the technique. The intuition is that $Q[fs, sl]$ is the aggregation of the rewards obtained when fs is used as fragment size together with sl as search limit. As α is constant, the updates perform exponential smoothing on the collected rewards. This allows to reduce more or less (depending on the value of α) the influence of early rewards. It takes the non-stationarity of the problem into account. This non-stationarity comes from the non-existence of a combination of fragment size and search limit that is effective from the beginning of the search until the end.

Parameters selection: Two different uses of the internal quantities to select the fragment size and the search limit are defined. The parameters are selected together as they are not independent.

1. Select randomly a pair (fs, sl) with probability of selection proportional to $Q[fs, sl]$
2. Select a pair (fs, sl) maximizing $Q[fs, sl]$

4.2 Informed Selection of Parameters

The heuristic described in this section also deals with the fragment size and the search limit parameters. It is called informed because it uses some knowledge about the parameters. As such, it is not directly an application of reinforcement learning.

The previous parameter selector uses only the information about whether an improvement has been made or not. This parameter selector uses, in addition to this information:

- the effective fragment size
- the effective number of failures
- number of non improving successive LNS steps
- whether the search tree defined by the fragment has been fully explored or not

where the effective fragment size is defined as the number of variables that have changed values when an improving solution is found. The effective number of failures is the number of failures to the improving solution, if one is found.

As internal quantities, this parameter selector only maintains and use one value for the current fragment size and one for the current search limit. The updates are based on the quantities previously mentioned. They are different for the fragment size and for the search limit. The search limit is expressed in number of failure nodes in the search tree.

Fragment size update: for current fragment size fs:

- if($improvement$) :
  $fs := fs + \alpha * (\beta * usefulFS - fs)$
- elseif($\#stagn \geq T$) :
  $fs := \gamma * fs$

where $usefulFS$ is the useful fragment size of the previous search and $\#stagn$ represents the number of consecutive non improving LNS steps. The constants α, β, γ and T are parameters of the technique. If an improvement has been made, the update narrows the fragment size in order to be close to the useful fragment size. The hope is that further improvements will involve a number of variables close to the useful fragment size of the current improvement. The parameter $\beta \geq 1$ allows the size of the fragment to be greater than the useful fragment size. After T unsuccessful LNS steps, the fragment size is enlarged by

a factor $\gamma > 1$. The hope is that this larger fragment size will allow bigger moves to find improving solutions. The fragment size is kept between bounds given as parameters to the technique.

Search limit update: for current search limit sl:

- if($improvement$) :
  $sl := sl + \alpha * (\beta * usefulSL - sl)$
- elseif($\neg treeFullyExplored$) :
  $sl := \gamma * sl$

where *usefulSL* represents the useful number of failures of the previous search and *treeFullyExplored* is true if the previous search tree has been fully explored. In case of an improvement, the search limit is narrowed to be close to the useful search limit. The hope is again that further improvements will involve a number of failures close to this useful one. $\beta \geq 1$ allows the current number of failure to be larger than the useful one. In case of non-improvement, if the search tree has not been fully explored, the search limit is enlarged by a factor $\gamma > 1$ in order to explore more of the further search spaces. The search limit is kept between bounds given as parameters to the technique.

4.3 Preliminary Computational Results

This section details the preliminary results obtained on the relaxed car sequencing problem. This problem has been chosen because LNS is known to be good at it (see for instance [1, 7]).

The relaxed car sequencing problem: the car sequencing problem is a constraint satisfaction problem. The goal is to sequence cars to be produced on an assembly line. The cars to be produced have to meet the demand. The demand consists of specific configurations of options. Each option is installed by a specific machine on the production line. Those have capacity constraints. For instance, a particular machine can only install its option on at most 3 cars in any sequence of 5 cars. The relaxation of this problem consists of inserting holes in the sequence. The objective is then to minimize the number of inserted holes without violating the capacity constraints of the machines. The objective function used is the total number of slots used to sequence all the cars.

Experimental setting: the instances of the car sequencing problem are the 30 instances from the CSPLIB[3] (C. Gagne's set). This set contains 10 instances with 200 cars to sequence, 10 instances with 300 cars and 10 instances with 400 cars. For each instance, 5 tests per technique are conducted. The mean of the objective value obtained with a time limit of 3 minutes is used in the comparisons. The means of the per instance standard deviations are also reported. Tests are done on an Intel Xeon 2.53GHz.

[3] http://csplib.org

Reference techniques: to test the effectiveness of the described techniques, the comparison is done with respect to tree other heuristics. The name of the techniques contains three identifiers: one for each LNS parameter. The first one corresponds to the size of the fragments, the second one to the search limit and the last one, to the fragment selection procedure. In those names, R means uniformly at random, F means fixed to a value experimentally known to be good for that problem (two first parameters), and S means a specific fragment selection procedure (third parameter), known to be good for that problem.

- *RRR:* at each LNS step, the fragment size, as percentage of the total number of variables, is selected randomly in the range [2,5,10,20,30,40,50]. The search limit, is selected randomly in the range [10,20,30,40,50,60,70,80, 90,100]. The variables to relax are selected uniformly at random.
- *FFR:* this parameter selector uses a fixed fragment size of 20 percents of the total number of variables and a fixed search limit of 50 failures. Those parameters have experimentally be found to be good for the car sequencing. The variables to relax are selected uniformly at random.
- *FFS:* this parameter selector uses the same fixed fragment size and search limit as the random fragment selector. It uses the car sequencing specific fragment selector described in [7]. It is specific in the sense that it uses knowledge on the car sequencing problem. It is only applicable to this particular problem.

The uninformed and informed selectors:

- *UUR:* this selector is the uninformed selector of section 4.1. Its settings are $\alpha = 0.1$ and the selection of the pair of parameters (fs, sl) maximizing $Q[fs, sl]$ as fragment size and search limit. The variables to relax are chosen uniformly at random.
- *IIR:* this selector is the informed selector of section 4.2. Its settings are $\alpha = 0.1$, $\beta = 1.5$ for the fragment size and 1.1 for the search limit, $\gamma = 1.1$ and the threshold T is 20. The fragment size bounds are 2 percents of the total number of variables and 50 percent. The bounds for the search limit are 10 failures and 100 failures. The variables to relax are chosen uniformly at random.

The mean number of slots used in the assembly line to sequence all the cars (left) and the mean of the per instance standard deviations (right) are reported in Table 1. The best results are shown in bold. As expected, the specific heuristic is the best on the three categories of problems. We can also see that the informed parameter selector is able to discover fragment size and failure limit equivalently good as the fixed values of the *FFR* selector. For the random selector, the parameters are fixed a priori to values experimentally known to be good for that problem.

#cars	*RRR*	*FFR*	*FFS*	*UUR*	*IIR*
200	209.5 — 0.8	210.7 — 1.1	**208.0** — 0.7	209.1 — 0.8	209.8 — 0.6
300	321.4 — 1.8	320.9 — 1.9	**314.9** — 2.0	320.6 — 2.0	320.8 — 2.3
400	428.5 — 2.1	424.7 — 2.7	**415.8** — 2.0	428.1 — 1.9	425.8 — 2.1

Table 1. Results on the car sequencing for the choice of fragment size and search limit

5 Fragment Selectors

The previous section deals with the first two parameters of the LNS. This section presents three heuristics for the last parameter of LNS: the fragment selection procedure. They differ in the quantities maintained, the rewards, the updates and/or the selection of the fragments. The section starts by presenting an inclusion quality based fragment selector. It then describes a variable quality based one and finally, presents a relation quality based selector. The names of the selectors come from the basic information they consider. Each section presents the results of its selector.

5.1 Inclusion Quality Based Fragment Selection

This technique performs an action for each variable independently. For each variable, the two possible actions are inclusion in the fragment and non-inclusion. Choosing one action for every variable results in the selection of a fragment. The goal is to detect and include variables that are important for optimizing the objective function.

Internal Quantities: for each variable x, this selector maintains:

$$Q[x, 0]\text{: quality of non-selection of } x$$
$$Q[x, 1]\text{: quality of selection of } x$$

Those quantities are positive during the entire search.

Rewards: the rewards are computed with respect to the output of the CP search:

$$reward = \begin{cases} 50 + 0.01 * nbIt & \text{if objective improved} \\ -1 - 0.01 * nbIt & \text{else} \end{cases} \quad (4)$$

where $nbIt$ represents the number of LNS iterations. In case of non-improvement, the reward is a penalty. Adding a fraction of the number of LNS iterations gives more importance to the rewards at the end of the search. Improving the objective at the end of the search is more difficult than at the beginning.

Updates: for each variable x of the previously selected fragment, the quantities are updated with the following rules: On improvement, increase the quality of selection:

$$Q[x,1] = Q[x,1] + \alpha * (reward - Q[x,1]) \tag{5}$$

where α is a constant parameter of the technique. This increase is only applied to variables that have changed value. In case of non-improvement, increase the quality of non-selection:

$$Q[x,0] = Q[x,0] + \alpha * (-reward - Q[x,0]) \tag{6}$$

The intuition behind those update rules is that they gather the rewards as quality of selection and the penalties as quality of non-selection. The penalties are made positives to keep the quality of non-selection positive. As the parameter α is constant, those updates perform exponential smoothing on the rewards gathered. This allows to vary (depending on the value of α) the influence of early rewards, taking the non-stationarity of the problem into account. The selection of fragment is a non-stationary problem because the good fragments to relax do not converge to a fixed fragment.

Fragment Selection: the fragment selection uses the following probability of selection (versus non-selection) for each variable x:

$$P[x] = \frac{Q[x,1]}{Q[x,0] + Q[x,1]} \tag{7}$$

For each variable x independently, it is included in the fragment with probability $P[x]$. The size of the fragment is automatically adjusted. If too large fragments are selected, leading to non-improvement of the objective, the probability of selection of all its variable decreases. This reduces the expected future fragment size.

Preliminary experimental results: As the selector of this section only deals with the selection of the fragments, the search limit parameter of LNS is fixed to a value experimentally good: 50 failure. Recall that the fragment size is automatically adjusted. Its name is *AFA*. The parameter α is set to 0.1. It is compared to *RRR*, *FFR* and *FFS* described in Section 4.3. For those, the experimental settings are the same as the ones described in Section 4.3.

The results for the inclusion quality based fragment selector are shown in Table 2. The mean value of the objective function is on the left and the mean of the per instance standard deviations is on the right. The best results are in bold. As we can see, this selector is not effective on this problem; it does not improve the pure random approach *RRR*. Moreover, its results are more variable.

5.2 Variable Quality Based Fragment Selection

This section defines variable quality based selectors. For this selector, a quantity (the quality of the variable) is maintained for each variable. It tries to capture

#cars	*RRR*	*FFR*	*FFS*	*AFA*
200	209.5 — 0.8	210.7 — 1.1	**208.0** — 0.7	214.7 — 2.3
300	321.4 — 1.8	320.9 — 1.9	**314.9** — 2.0	329.3 — 3.4
400	428.5 — 2.1	424.7 — 2.7	**415.8** — 2.0	434.0 — 2.6

Table 2. Results of the inclusion quality based fragment selector on the car sequencing

the importance of the variable for the objective function. The goal is to select fragments composed of variables that will allow a reoptimization of the objective function. Two selectors are presented. They differ in the utilization of the internal quantities.

Internal Quantities: for each variable x, this selector maintains:

$$V[x]: \text{ the quality of the variable } x \tag{8}$$

Those quantities are positive during the entire search.

Rewards: the rewards depend on the output of the CP search:

$$\mathit{reward} = \begin{cases} 50 & \text{if objective improved} \\ 0 & \text{else} \end{cases} \tag{9}$$

There is no need to add a fraction of the number of iterations here. The quality of a variable is relative to the quality of the other variables.

Updates: both when improvement and non-improvement, the update rule for each variable x of the previous fragment is the following.

$$V[x] = V[x] + \alpha * (\mathit{reward} - V[x]) \tag{10}$$

where α is a constant parameter of the technique. The intuition behind this update rule is again the collection of the rewards. As α is constant, the updates perform exponential smoothing. This reduces the influence of early rewards. When improvement, the reward is only granted to variables that have changed value.

Fragment Selection: Two different fragment selectors are defined. They Both select variables until a given fragment size is attained. They can be described as:

1. Iteratively select a variable x among available ones with probability of being selected proportional to $V[x]$
2. Include in the fragment variables with the highest quality V

Preliminary experimental results: as in Section 5.1, the variable quality based selector is compared to *FFR* and *FFS* within the same settings. For the value based selector, the fragment size is fixed to 20% of the number of variables and the search limit is 50 failures. It is thus called *FFV*. In those tests, α is set to 0.1 and the selection of the fragments corresponds to the selection scheme 1.

The results of the variable quality based selector are shown in Table 3. The mean value of the objective function lies on the left and the mean of the per instance standard deviations lies on the right. Again, the best results are in bold. As we can see, this selector is slightly less performant on this problem than *FFR* where the selection of fragments is done randomly.

#cars	*FFR*	*FFS*	*FFV*
200	210.7 — 1.1	**208.0** — 0.7	210.6 — 1.2
300	320.9 — 1.9	**314.9** — 2.0	322.7 — 2.2
400	424.7 — 2.7	**415.8** — 2.0	428.5 — 2.5

Table 3. Results of the variable quality based selector on the car sequencing

5.3 Relation Quality Based Fragment Selection

This section presents various relation quality based selectors. For each pair of variables, they maintain a quantity representing the quality of the relation between each pair of variables. The goal is to be able to select fragments composed of closely related variables. All the selectors of this section share the same internal quantities. Although, two different reward systems and update rules are defined and there are five different utilizations of the quantities for each reward-update.

Internal Quantities: for each pair of variables (x, y), this selector maintains

$$Q[x, y] : \text{ the quality of the relation between } x \text{ and } y \tag{11}$$

Those quantities try to capture the importance of a variable for another in a fragment. For instance, if two variables are linked by a strong constraint, it is important to relax them simultaneously to allow them to change value. The quality of the relation between the two variables should then be high. The matrix Q is kept symmetric and all its entries are positive during the search.

Rewards: two different reward systems are defined:

$$reward_1 = \begin{cases} 50 & \text{if objective improved} \\ 0 & \text{else} \end{cases} \tag{12}$$

and

$$reward_2 = \begin{cases} 50 & \text{if objective improved} \\ -10 & \text{else} \end{cases} \tag{13}$$

In the second reward system, the reward corresponds to a penalty when no improvement has been made.

Updates: two update rules are defined. The first uses exponential smoothing with the first reward system. The second uses the second reward system without exponential smoothing. They are detailed hereafter. They are applied for each pair of variables (x, y) of the previously selected fragment.

With exponential smoothing:

$$Q[x, y] = Q[x, y] + \alpha * (reward_1 - Q[x, y]) \tag{14}$$

where α is a constant parameter of the technique. Without exponential smoothing:

$$Q[x, y] = Q[x, y] + reward_2 \tag{15}$$

For this last update rule, the positivity of the elements of Q is restored after each update.

The intuition behind those updates is again the aggregation of the rewards. In both cases, when an improvement is made, the reward is only granted between pairs of variables that have changed value.

Fragment Selection: five fragment selection techniques are defined. Each of them can be used with any update rule. They all start by picking a random variable and iteratively select one variable at a time until a given fragment size is attained. Those fragment selections are detailed hereafter:

1. Select variable x among available ones with probability of being selected proportional to

$$\sum_{y \in fragm} Q[y, x] \tag{16}$$

2. Select variable x among available ones with maximum

$$\sum_{y \in fragm} Q[y, x] \tag{17}$$

3. Select variable randomly among the 10 variables maximizing the quantity

$$S[x] = \sum_{y \in fragm} Q[y, x] \tag{18}$$

4. Let $lastVar$ be the last variable included in the fragment. Select variable randomly among the 10 variables maximizing $S[x] = Q[lastVar, x]$
5. Let $lastVar$ be the last variable included in the fragment. Select variable among the 10 variables maximizing $S[x] = Q[lastVar, x]$ with probability of being selected for x proportional to

$$\sum_{y \in fragm} Q[y, x] \tag{19}$$

All those selectors are targeted at selecting fragments with related variables. Such fragments allow large search spaces for the CP search.

Preliminary experimental results: the results of two different settings for this selector are presented in Table 4. As in the previous sections, the mean of the objective value lies on the left and the mean of the per instance standard deviations lies on the right.

The first one ($FFRel_1$) uses the first update rule (exponential smoothing) and selects fragments using the selection scheme 1. It uses the same fixed values for the fragment size and search limit as the ones used in FFR and FFS: 20% of the number of variables are relaxed. The search limit is 50 failures. The parameter α is set to 0.1.

The second setting ($FFRel_2$) uses the second update rule and the same fixed parameters as in the first setting. Both settings of relation quality based selector are compared to the same techniques as in the previous sections.

As we can see in Table 4, the selector using exponential smoothing ($FFRel_1$) is slightly more effective than its unsmoothed version ($FFRel_2$) but also slightly more variable. However, relation quality based selectors are not better than the random fragment selector with fixed fragment size and search limit on this problem.

#cars	FFR	FFS	$FFRel_1$	$FFRel_2$
200	210.7 — 1.1	**208.0** — 0.7	210.0 — 0.9	210.6 — 0.9
300	320.9 — 1.9	**314.9** — 2.0	320.0 — 2.5	321.1 — 2.2
400	424.7 — 2.7	**415.8** — 2.0	425.5 — 2.3	425.9 — 2.1

Table 4. Results of two relation quality based selectors on the car sequencing

6 State of the Art Generic Fragment Selector

The fragment selectors described in the previous section are not very effective compared to a random selection of the fragments. To put those results into perspective, a comparison of the best fragment selector developed in this work with the state of the art generic fragment selection heuristic [1] is made.

The fragment selection heuristic described in [1] is called Propagation Guided Large Neighborhood Search (PGLNS). PGLNS is using information raised by the propagation to select the fragments. The goal is to select fragments containing related variables. In a nutshell, variables are considered as related if a lot of propagation occurs between the variables when one is fixed. This propagation is measured during the assignation of the non-fragment variables. This implies that the variables are to be fixed one by one and that, each time, the fixed point of propagation must be reached. In contrast, for all the techniques presented in this paper (including the random techniques and the specific ones) all the non-fragment variables are fixed together. This allows to reduce dramatically the time to relax a fragment.

The results of the Comet reimplementation of [1] (named *FFPglns*) is presented in Table 5. As in the previous sections, the mean of the objective value lies on the left and the mean of the per instance standard deviations lies on the right. For the tests, the size of the fragments and the search limit is fixed to the same values as for *FFR* and *FFS*. The relaxation process of PGLNS is significantly slower. Within the same time limit, the random heuristic performs in average 20 times more iterations than PGLNS.

#cars	*FFR*	*FFS*	$FFRel_1$	*FFPglns*
200	210.7 — 1.1	**208.0** — 0.7	210.0 — 0.9	211.3 — 1.1
300	320.9 — 1.9	**314.9** — 2.0	320.0 — 2.5	335.4 — 2.8
400	424.7 — 2.7	**415.8** — 2.0	425.5 — 2.3	443.0 — 5.2

Table 5. Results of the reimplementation of PGLNS on the car sequencing

As we can see, our $FFRel_1$ heuristic for fragment selection provides better and less variable results than PGLNS on this problem.

7 Conclusion

This paper presents preliminary results about heuristics to define the three LNS parameters. Two of them select the size of the fragment and the CP search limit. Three provide methods for the selection of the fragments. These heuristics use the Reinforcement Learning framework. Preliminary results are shown on a relaxation of the car sequencing problem. These results show that the selection of the fragment size and the search limit are more effective than a pure random choice for those parameters. The heuristics for selecting the fragment, however, do not perform better than the standard random fragment selection with fixed good fragment size and search limit. The heuristics are shown to be more effective than the state of the art generic method on the car sequencing problem.

How positive are these preliminary results? The results suggest that the proposed reinforcement learning heuristic is able to discover the best parameters for

the fragment size and the search limit. However, for the fragment selection procedure, the results suggest that the proposed reinforcement learning heuristics are not more effective than a random selection. One should notice that the only information used by these heuristics is whether an improvement was made or not. Generic heuristics for LNS could include, in addition to the improvement, information from the search, such as propagation information to catch the links between variables, as in PGLNS. This information could also include the influence of the bindings of the variables on the objective function bounds, etc. Combining different measures made during the search within the reinforcement learning framework presented in this paper may be a way to obtain a better generic Large Neighborhood Search.

Acknowledgment: This research is partially supported by the Interuniversity Attraction Poles Programme (Belgian State, Belgian Science Policy) and the FRFC project 2.4504.10 of the Belgian FNRS.

References

1. Laurent Perron, Paul Shaw and Vincent Furnon. Propagation Guided Large Neighborhood Search. CP 2004, LNCS 3258 (2004) 468–481.
2. Daniel Godard, Philippe Laborie and Wim Nuijten. Randomized Large Neighborhood Search for Cumulative Scheduling. AAAI 2005.
3. Guy Desaulniers, Eric Prescott-Gagnon and Louis-Martin Rousseau. A Large Neighborhood Search Algorithm for the Vehicle Routing Problem with Time Windows. MIC 2007.
4. Emilie Danna and Laurent Perron. Structured vs. Unstructured Large Neighborhood Search : A Case Study on Job-Shop Scheduling Problems with Earliness and Tardiness Costs. CP 2003, LNCS 2833 (2003) 817–821.
5. Paul Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems CP 1998, LNCS 1520 (1998) 417–431.
6. Richard S. Sutton, Andrew G. Barto Reinforcement learning : an introduction Cambridge, MA: MIT Press, 1998
7. Laurent Perron and Paul Shaw. Combining Forces to Solve the Car Sequencing Problem CPAIOR 2004, LNCS 3011 (2004) 225-239.
8. P. Schaus, P. Van Hentenryck, J-N. Monette, C. Coffrin, L. Michel and Y. Deville Solving Steel Mill Slab Problems with constraint-based techniques: CP, LNS, and CBLS Constraints volume 16 (2011), Springer Netherlands, 125–147