Generic Adaptive Heuristics for Large Neighborhood Search

Jean-Baptiste Mairy[1], Pierre Schaus[2], and Yves Deville[1]

[1] Université Catholique de Louvain, Belgium
[2] Dynadec Europe, Belgium

Abstract. The Large Neighborhood Search (LNS) approach for solving Constrained Optimization Problems has been proved to be effective on a wide range of problems. LNS is a local search metaheuristic that uses a complete search method (such as CP) to explore a large neighborhood. At each step, the neighborhood is defined by relaxing a subset, called fragment, of the variables in the current solution. Despite the success of LNS, no general principle has emerged on how to choose the fragment from one restart to the other. Practitioners often prefer to relax randomly the solution to favor diversification. This work focuses on the design of generic adaptive heuristics for choosing automatically the fragment in LNS, improving the standard random choice. The defined heuristics are tested on the Car Sequencing problem for which we introduce a new original relaxation. From those experiments, we conclude that all our heuristics except one are performing better than a random fragment selection. We also show that our *mean dynamic impact proximity* and *min/max dynamic impact proximity* heuristics are significantly better than all the others.

1 Introduction

Finding optimal solutions to hard combinatorial optimization problems is one of the major challenges in computer sciences. Large Neighborhood Search (LNS) has been proved to be effective on many problems at quickly finding good solutions (see for instance [1, 6, 3–5]). The LNS approach takes advantage of the expressiveness of Constraint Programming (CP) and the speed of Local Search (LS). Its working principle is to maintain a *candidate solution* through the search that is *not* violating any constraint but that may not be optimal (or not known to be). The successive solutions are obtained by repeating the two following operations until a stopping criterion is met.

1. *neighborhood definition*: this step consists of choosing the set of variables (fragment) that will be relaxed to their original domains while fixing the other variables to their value in the current best solution. The domain of the relaxed variables defines the neighborhood of the current solution that will be explored with CP at the next step.

2. *neighborhood exploration*: this step consists of using CP to explore the restricted problem defined by the relaxation of the fragment. When an improved solution is found, the current best solution is replaced. A limit in time or in number of failures is specified to avoid spending too much time in exploring the neighborhood.

Although the LNS framework is really simple, its parameters (the size of the fragments, the fragments selection procedure and the limit on the CP exploration step) must be carefully chosen. No general principle has emerged yet on how to choose the fragment from one restart to the other. Practitioners often prefer to relax randomly the solution to favor diversification.

A self-adapting LNS for scheduling problems has been proposed in [8]. Three strategies are proposed for the selection of the fragment. At each step, a strategy is chosen depending of its efficiency in the previous LNS steps. The efficiency measure of the chosen strategy is then adapted depending on the quality of the improvement made by this LNS step on the best solution. This approach thus combines LNS with a portfolio of fragment selection algorithms. The strategies of the portfolio are specific to scheduling problems.

Generic heuristics for the fragment selection were introduced by Perron et al. [1]. The authors define two adaptive procedures. The first one starts from the original CSP (where the domains of the variables are their initial ones) and successively selects the variables that will be bound in the next LNS step. When a variable is selected, it is bound to its value in the current best solution. The propagation of this new binding is then realized. The next variable to be selected is the variable whose domain has been the most reduced by the propagation. The second heuristic selects the variables to be relaxed in order to define fragments with *close* variables. The *closeness* relation between variables is based on the mean amount of propagation that occurred when fixing the non-fragment variables in the LNS procedure.

In the context of Weighted CSP (WCSP), where the constraints can be violated at a certain cost, the objective is to minimize the total cost of the violations. The analogy for LNS in this context is the Variable Neighborhood Search (VNS). Levasseur et al. propose in [2] different methods, based on the constraint network and on the violation cost of the constraints, to choose the fragment of variables.

This paper pursues the work of Perron et al. bringing the following contributions:

- New *generic adaptive heuristics* dedicated to the choice of the fragments,
- An original relaxation and a CP model for the Car Sequencing problem taking advantage of softglobal constraints, and
- Experimental results showing the effectiveness of our heuristics on the Car Sequencing problem.

Outline. Section 2 introduces fragment selection strategies based on impact based quality measures of variables. The objective is to automatically discover fragments with a high potential improvement of the objective function. Section 3 experiments the strategies on a new relaxation of the car sequencing problem.

2 Generic Adaptive Heuristics for LNS

This section focuses on the different heuristics developed in the context of this work. We start by presenting the concepts and measures used for defining the heuristics. Next, we present simple heuristics using a single measure. Finally, combined heuristics, using more than one measure are presented.

2.1 Concepts and Measurements

Dynamic Impact. The *dynamic impact* of a variable tries to capture the impact that the variable would have on the objective if it is relaxed. Let S be a solution to the problem. The *impact of a value v for a variable x_i* is defined as the difference that is induced on the objective by setting x_i to v rather than to its value in S. While evaluating this impact, the variables other than x_i are kept fixed to their values in S and the constraints are ignored. That is, for an objective O, the impact of v for x_i on O with respect to S is

$$\mathcal{I}(x_i, v, S) = O\Big(S(x_1), \ldots, S(x_{i-1}), v, S(x_{i+1}), \ldots, S(x_n)\Big) - O\Big(S(x_1), \ldots, S(x_n)\Big) \tag{1}$$

where $S(x_k)$ is the value of the variable x_k in the solution S.

The *impact of a variable* x_i for a solution S is the aggregation of $\mathcal{I}(x_i, v, S)$ for all the values $v \in D_0(x_i)$ (initial domain of x_i) except for $S(x_i)$. Our dynamic impact is separated into two different measures corresponding to two different aggregations: the *mean dynamic impact* and the *min/max dynamic impact*.

Mean Dynamic Impact. Let x_i be a variable, O be the objective function of the problem and S be a solution. The *mean dynamic impact* $\mathcal{I}_{mean}$ of variable x_i with respect to S is defined as

$$\mathcal{I}_{mean}(x_i, S) = \frac{1}{|D_0(x_i)| - 1} \sum_{v \in D_0(x_i) \setminus \{S(x_i)\}} \mathcal{I}(x_i, v, S) \tag{2}$$

A variable that improves the objective on average when changing only its value has a positive $\mathcal{I}_{mean}$ if the problem is a maximization and a negative one if the problem is a minimization.

Max Dynamic Impact. For *maximization* problems, the *max dynamic impact* takes as impact for a variable x_i, the maximum value of $\mathcal{I}(x_i, v, S)$ for $v \in D_0(x_i)$ different from $S(x_i)$. It corresponds to the best improvement that the variable may induce on the objective by only changing its value. For a variable x_i and a solution S, the *max dynamic impact* is thus

$$\mathcal{I}_{max}(x_i, S) = \max_{v \in D_0(x_i) \setminus \{S(x_i)\}} \mathcal{I}(x_i, v, S) \tag{3}$$

This impact measure is handled analogously for *minimization* problems.

Adaptive Impact. The *Adaptive Impact* measure tries to capture the concept of the impact of a variable by recording information each time a variable is bound in the LNS process. The goal is to reflect the intrinsic importance of the variables for the objective, independently from the current solution.

The adaptive impact measures the influence that the bindings of the variables have on the bounds of the objective through propagation. For a maximization problem, the bounds that are monitored are the upper bounds. For a minimization, the monitored bounds are the lower bounds. The Adaptive Impact of a variable x_i is defined as the average change in the bound of interest through the bindings of x_i since the beginning of the whole LNS search. Suppose the LNS search is at a point of advancement p and that $SumImpact(x_i, p)$ is the sum of the changes in the bound of interest that occurred while binding x_i until p and that $NumTimesBound(x_i, p)$ represents the number of times x_i has been bound until p. The adaptive impact of x_i is then:

$$\mathcal{I}_{adaptive}(x_i, p) = \frac{SumImpact(x_i, p)}{NumTimesBound(x_i, p)} \tag{4}$$

The different definitions of the impact of the variables handle each variable independently of the others. Further definitions may include variable interactions.

Variability. During the LNS search process, when a variable is relaxed, the CP search may reassign it to the same value as in the current best solution. This means that including this variable in the fragment was pointless. The concept of *variability* is defined as the likelihood of a variable to change value in an improving solution if it is relaxed.

The measure of the variability is based on the successive solutions exhibited during the search. It is defined as the ratio between the number of different values that a variable has taken in the solutions found and the number of times the variable has been relaxed. Suppose the LNS search is at a certain point p where $\mathbf{S}(p)$ is the set of solutions found until p and $NumRelax(x_i, p)$ is the number of times x_i has been relaxed until p. The variability of x_i at point p is

$$var(x_i, p) = \frac{\left|\{v \mid \exists s \in \mathbf{S}(p) : s(x_i) = v\}\right|}{NumRelax(x_i, p)} , \tag{5}$$

where $|\{\dots\}|$ is the cardinality of the set $\{\dots\}$.

Proximity. The *proximity* is dealing with the relationships between variables and was inspired by the *closeness* measure defined in [1]. It is defined in terms of the global links that the entire constraint network induces between pairs of variables. The measure is made by inspecting the propagation occurring when variables are bound. Practically, the proximity between variables x_i and x_j is computed as the average number of values that are removed from $D(x_j)$ when

x_i is bound. The only subtlety is that this average is computed only over the bindings of x_i where x_j is not bound. Otherwise, the computation would include cases where there is no propagation only because x_j is bound.

Suppose the LNS search is at a point p, where $Propag(x_i, x_j, p)$ is the *total* number of values removed from $D(x_j)$ through all the bindings of x_i until p. Let $NumBinds(x_i, x_j, p)$ be the number of times x_i has been bound while x_j was not bound until p. The proximity between x_i and x_j is then

$$Proximity(x_i, x_j, p) = \frac{Propag(x_i, x_j, p)}{NumBinds(x_i, x_j, p)} \tag{6}$$

If two variables have a large proximity, relaxing the second one and not the first one would directly prune a large number of values in the relaxed variable domain. This would lead to few opportunities for the relaxed variable to change value. Note that the proximity relation is not symmetric.

The proximity measure is based on the same intuition as the closeness relation defined in [1]: the relations between variables can be revealed by the propagation. However, the *proximity* is updated *each time* a variable is bound. This includes the fragment freezing and the CP solving steps. The closeness relation defined in [1] is only updated during the freezing of the fragments. Moreover, as we will see, we only use the *proximity* in addition to other heuristics, which is not the case in [1].

2.2 Simple Heuristics

This section presents heuristics built from one of the measures previously presented.

Impact Based Heuristics. These heuristics aimed at finding a fragment composed of variables with large impacts on the objective. Some randomization is also introduced for the sake of diversification. This randomization is provided by selecting variables to relax among the best impact variables.

The pseudo-code algorithm 1.1 shows the fragment selection process. In this code, X refers to the set of variables of the problem, d refers to the desired fragment size, $\mathcal{I}_\lambda$ is the chosen impact measure, $\oplus$ and $\ominus$ are respectively the addition and removal of an element from a list. The selection of a variable is a random selection where the probability for a variable to be selected is proportional to its impact value $\mathcal{I}_\lambda$.

Variability Based Heuristic. When the variability measure is used to select the fragment, the goal is to select variables that have a large variability since a large variability indicates that the variable is likely to change value in an improving solution. The diversification is increased by the same mechanism as the one used in the impact based heuristics. Pseudo-code algorithm 1.2 shows the selection of a fragment based on the variability.

$fragment = \{\}$
$varSet = X$
while $(|fragment| < d)$ **do**
  Select variable x in $varSet$ according to $\mathcal{I}_\lambda$
  $fragment \oplus x$
  $varSet \ominus x$
end while

Algorithm 1.1 : Fragment selection procedure for impact based heuristics

$fragment = \{\}$
$varSet = X$
while $(|fragment| < d)$ **do**
  Select variable x among maximum var in $varSet$
  $fragment \oplus x$
  $varSet \ominus x$
end while

Algorithm 1.2 : Fragment selection procedure for variability based heuristics

2.3 Combined Heuristics

This section proposes *combined heuristics*. The combination is made by combining an *impact* measure with either the *variability* or the *proximity* measure.

Impact + Variability. Combining an impact measure with the variability measure aims at selecting fragments containing important variables that are likely to change values in an improving solution. The two measures are here considered as equally pertinent. Since the variability of any variable lies between 0 and 1, the impacts measures are normalized to lie between 0 and 1 as well. This expression depends on the impact measure as well as the type of problem (minimization or maximization).

– When the chosen impact measure $\mathcal{I}_\lambda$ is the *mean dynamic impact* or the *min/max dynamic impact* and the problem is a *maximization*, the *normalized* impacts are:

$$\widehat{\mathcal{I}}_\lambda(x_i) = \begin{cases} \max\left(\frac{\mathcal{I}_\lambda(x_i)}{\max_{x_k}(\mathcal{I}_\lambda(x_k))}, 0\right) & \text{if } \max_{x_k}(\mathcal{I}_\lambda(x_k)) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

– For the same impact measures $\mathcal{I}_\lambda$ but for a *minimization* problem, the *normalized* impacts are:

$$\widehat{\mathcal{I}}_\lambda(x_i) = \begin{cases} \max\left(\frac{\mathcal{I}_\lambda(x_i)}{\min_{x_k}(\mathcal{I}_\lambda(x_k))}, 0\right) & \text{if } \min_{x_k}(\mathcal{I}_\lambda(x_k)) < 0 \\ \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

– When the chosen impact measure is the *adaptive impact*, then, regardless of the type of problem, the *normalized* impacts are:

$$\widehat{\mathcal{I}}_{adaptive}(x_i) = \begin{cases} \frac{\mathcal{I}_{adaptive}(x_i)}{\max_{x_k}(\mathcal{I}_{adaptive}(x_k))} & \text{if } \max_{x_k}(\mathcal{I}_{adaptive}(x_k)) > 0 \\ \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Those normalizations correspond to the ratio between the impact of the variable and the *best* impact.

The values that are used by the impact-variability heuristics to select the variables in the fragment are the sums of the normalized impacts and the variabilities. For a variable x_i, the value $\mathcal{IV}_\lambda(x_i)$ used by the heuristics based on impact measure $\mathcal{I}_\lambda$ is defined as:

$$\mathcal{IV}_\lambda(x_i) = \widehat{\mathcal{I}}_\lambda(x_i) + var(x_i) \tag{10}$$

The procedure for selecting the fragment based on $\mathcal{IV}_\lambda$ is the same as the procedure shown in algorithm 1.1 but using $\mathcal{IV}_\lambda$ instead of $\mathcal{I}_\lambda$. The goal of the selection is to include variables with *large* values of $\mathcal{IV}_\lambda$ in the fragment.

Another combination of those two measures would use a product instead of a sum. This aggregation would force the selected variables to have good values for *both* measures. However, this aggregation is not explored in this paper.

Impact and Proximity. When combining an impact measure with the proximity one, the fragment is treated as being composed of two parts with distinct functionality. The proposed heuristics try to select fragments composed of *important* variables together with variables that are *linked* with the important ones. This second part of the fragment allows important variables to change values and improve the current solution.

The impact and the proximity are here treated as equally relevant. Each part of the fragment thus receives half of the total number of variables in the fragment. The second half of the fragment is selected in order to contain variables linked with a large number of variables from the first half. To determine if a variable is linked or not with another one, a threshold on the proximity is used. A variable x_i is considered as linked with another variable x_j if $proximity(x_i, x_j)$ is greater than the threshold. Since the proximity is not symmetric, it is considered both ways. For a variable x_i, a first half fragment F_1 and a threshold T, its inclusion in the second half of the fragment is based on the number of variables from F_1 to which x_i is linked plus the number of variables from F_1 that are linked to x_i:

$$NL(x_i, F_1, T) = \big|\{x_j \in F_1 \mid proximity(x_i, x_j) > T\}\big| + \big|\{x_j \in F_1 \mid proximity(x_j, x_i) > T\}\big| \quad (11)$$

The pseudo-code algorithm 1.3 shows how the fragment is selected by the heuristics of this section. In this code, $\mathcal{I}_\lambda$ is the chosen *impact* measure. The selection of the first half F_1 is the same as the one presented for *impact based* heuristics except for the number of selected variables. The diversification is provided in both fragment parts by the same randomization mechanism as for the previous heuristics.

```
F1 = {}
varSet = X
while (|F1| < d/2) do
   Select variable x in varSet according to I_λ
   F1 ⊕ x
   varSet ⊖ x
end while
F2 = {}
while (|F2| < d/2) do
   Select variable x in varSet among large NL(x, F1, T)
   F2 ⊕ x
   varSet ⊖ x
end while
fragment = F1 ∪ F2
```

Algorithm 1.3 : Fragment selection procedure for impact-proximity heuristics

3 Experimental Results

3.1 The Car Sequencing as an Optimization Problem

The Car Sequencing (CS) problem consists of sequencing cars on a production line. A set of *demands* for each *configuration* of cars to be produced is specified. A configuration is composed of specific *options* (e.g. sunroof, air-conditioning, etc.). Each option o is installed by a capacited *station*: at most m_o out of n_o consecutive cars can have it. The goal of this problem is to arrange the cars in a sequence such that the station capacities are not exceeded.

Since the car sequencing is a feasibility problem and that LNS is intended for optimization problem, we relax it: multiple cars from the same configuration can be located on the same slot. The objective of the relaxed problem becomes to maximize the number of slots used. A feasible solution to the original problem is found when the number of slots used is equal to the number of cars. This relaxation differs from the usual relaxation (which minimizes the number of

violations of the capacity constraints) because the LNS only considers satisfying solutions.

The *Comet* model for the relaxation is presented on Figure 1. The data of the problem are `demand`, the number of cars for each configuration to place, `m[o]` and `n[o]` the capacity constraints of each option `o`, `configOfOption` is the array containing for each option, the set of configurations requiring it, `carsWithConfig` is an array containing for each configuration, the set of cars with that configuration. The decision variables are `placeOfCar`, the *slot* where each car is placed. The variables `configOfCar` represent the configuration coming in each slot. This is another viewpoint introduced only to post the `sequence` constraint. The two viewpoints (variables `placeOfCar` and `configOfCar`) are linked with element constraints. Note that as multiple cars are allowed on a slot, when two cars are on the same slot, they are treated as being one car. Note also that an *empty* slot (i.e. a slot not present in `placeOfCars`) can contain any car configuration that is *compatible* with the surrounding slots. This constraint allows the search tree to be further pruned. We thus think that this relaxation is *stronger* than the one proposed in [1]. In this relaxation, additional slots and additional virtual cars requiring no option are introduced. The objective is then to minimize the last slot with a real car. Our relaxation also make use of the soft global constraint *atLeastNValue* [7] which is a soft version of the well known *allDifferent* constraint. This constraint links globally the objective (the number of different slots with a car) and the decision variables allowing a strong filtering. Another advantage is that we can use the *maximal matching* maintained by the filtering of *atLeastNValue* as a value heuristic to assign the cars to the slots. This is achieved in the using block in the left alternative. The drawback of this model is that it introduces symmetries between cars of the same configuration. These symmetries are broken dynamically during the search in the right alternative of a choicepoint: when entering the right alternative, it is proved that a slot is not possible for the car, so this slot is also removed from all the not yet assigned cars of the same configuration. Whenever the limit on the number of fails is reached, the `onRestart` block is executed, a fragment is chosen and variables not in this fragment are fixed to their value in the current best solution.

3.2 Experimental settings for comparing fragment selection strategies

The instances used for the tests are instances of the CSPLIB[1]. We have chosen to use the instances of the second set proposed in CSPLIB. All those instances are satisfiable, which means for each one that there exists a solution for which the number of nonempty slots equals the number of cars.

The test procedure has been designed to evaluate the fragment selection heuristics *independently* of the other LNS parameters. It is based on the comparison of the results of the strategies over *one* LNS step to reduce the influence of the other LNS parameters. In order to be as independent as possible from

[1] http://www.csplib.org/

```
//Data of the problem
range Options; range Configs; range Slots; range Cars;
int m[Options]; int n[Options]; //capacity m[o] out of n[o] for option o
int requires[Configs,Options];  //1 if config requires option, 0 otherwise
int demand[Configs]; //demand for each configuration c
int config[Cars]; //configuration for each car
set{int} configsOfOption[Options]; //set of cars with option
set{int} carsWithConfig[Configs]; //set of cars with config


Solver<CP> cp();
var<CP>{int} configOfCar[Slots](cp,Configs);//config of the car in the slot
var<CP>{int} placeOfCar[Cars](cp,Slots);//slot of the car
var<CP>{int} atLeast(cp,Cars);//number of slots with a car

AtLeastNValue<CP> atLeastCons;

cp.lnsOnFailure(3200); //LNS Limit

maximize<cp>
  atLeast
subject to{
  forall(o in Options)
    cp.post(sequence(configOfCar,demand,m[o],n[o],configsOfOption[o]));
  forall(c in Cars)
    cp.post(configOfCar[placeOfCar[c]]==config[c]);
  atLeastCons = atLeastNValue(placeOfCar,atLeast);
  cp.post(atLeastCons);
}
using{ //Search Heuristic
 while(!bound(placeOfCar))
    selectMin(c in Cars: !placeOfCar[c].bound())(placeOfCar[c].getSize()){
      int val = placeOfCar[c].getMin();
      if(atLeastCons.hasValInBestAssignment(c))
         val = atLeastCons.getValInBestAssignment(c);
      try<cp>{
         cp.post(placeOfCar[c]==val);
      }|{
        forall(c_ in carsWithConfig[config[c]]:!placeOfCar[c_].bound())
         cp.post(placeOfCar[c_]!=val);
      }
   }
}onRestart{ //LNS Relaxation
   set{int} relaxedCars();
   //.. fragment definition ..
   Solution sol = s.getSolution();
   forall(c in Cars:!relaxedCars.contains(c))
      s.post(placeOfCar[c] == placeOfCar[c].getSnapshot(sol));
}
```

Fig. 1. Model of the relaxed Car Sequencing problem

these parameters, each heuristic has been tested with its best parameters. The test procedure can be summarized as follows

1. For each instance, compute five *starting solutions* by running the strategies for a fixed number of steps (the same number for all the solutions). Those solutions, in addition to the values of the variables, contain all the quantities that are needed by the adaptive measures.
2. Based on the starting solutions, evaluate, for each strategy, the optimal fragment size and failure limit parameters.
3. From each starting solution, run multiple times each strategy with the optimal parameters for a unique LNS step.

For the optimal parameters evaluation, fragment sizes of 25, 20, 15, 10, 5, 2 percents of the total number of variables are tested with a limit of 3200 backtracks for the CP solving step (this parameter can be fixed a priory since the comparison is based on one LNS step). The optimal fragment size for a strategy is the one leading to the largest number of improvements when running the process 10 times for each starting solution.

The evaluation of the strategies is done by running them for one LNS step from the starting solutions with the optimal fragment size. Each strategy is run 30 times per instance (6 times from each starting solution). The comparison of the different strategies is based on the following collected statistics:

- the number of improvements
- the average improvement
- the average execution time

These are reported for each strategy/instance, except the average execution time which is reported per strategy.

3.3 Computational Results

In the car sequencing problem, one can show that the *mean dynamic impact* and the *max dynamic impact* lead to the same fragment selections. The objective to be maximized is the number of different values and the variables represent the slot assigned to each vehicle. Hence the max dynamic impact is proportional to the mean dynamic impact.

The starting solutions were obtained with a limit of 35 LNS steps. Table 1 shows the optimal fragment sizes obtained with the test procedure. Table 2 presents the number of improvements and Table 3, the average improvements. In the last two tables, the best results are highlighted in bold. The mapping between the strategies and the names of the columns is the following:

- rand → *random neighborhood*
- AI → *adaptive impact*
- AIP → *adaptive impact + proximity*
- AIV → *adaptive impact + variability*

– MDI → *mean dynamic impact*
– MDIP → *mean dynamic impact + proximity*
– MDIV → *mean dynamic impact + variability*
– Var → *variability*

Heuristic	rand	AI	AIP	AIV	MDI	MDIP	MDIV	Var
Fragment size (%)	15	10	15	10	15	15	10	15

Table 1. Optimal fragment sizes in percentage of the number of variables

instance	rand	AI	AIP	AIV	MDI	MDIP	MDIV	Var
bench_65_01	4	11	8	9	12	**13**	**13**	7
bench_65_03	11	14	10	8	**24**	19	14	9
bench_70_01	1	5	6	0	4	**12**	7	2
bench_70_03	3	4	5	4	**12**	10	3	5
bench_70_04	6	16	13	6	18	**24**	9	10
bench_75_01	3	**10**	5	9	6	6	6	5
bench_75_02	8	9	13	9	12	**15**	12	4
bench_75_03	8	13	13	6	6	**24**	10	9
bench_75_04	7	10	6	4	**24**	23	12	4
bench_80_01	8	**19**	13	3	6	**19**	11	5
bench_80_02	9	10	10	9	**23**	18	16	12
bench_80_03	6	11	11	5	18	**19**	8	6
bench_80_04	2	5	**14**	0	8	12	2	4
bench_85_01	6	6	7	4	**13**	11	**13**	4
bench_85_02	11	15	12	11	15	**17**	11	8
bench_85_04	8	**21**	7	6	18	8	14	9
bench_90_01	4	17	7	2	**26**	21	16	13
bench_90_02	2	10	7	2	**21**	17	8	5
bench_90_03	4	5	11	1	**19**	17	3	7
sum	111	211	178	98	285	**305**	188	128

Table 2. Number of improvements on the Car Sequencing problem

From those Tables, the first conclusion is that our heuristics outperform the random neighborhood heuristic, except for the *adaptive impact + variability* (AIV). The random approach introduces probably too much diversification.

The best heuristics on this problem are the *mean dynamic impact + proximity* (MDIP) followed by the *mean dynamic impact* (MDI). This can be justified by the right level of diversification induced by MDI. Indeed, it is more optimistic than the others, leading to a better diversification of the solutions. The

instance	rand	AI	AIP	AIV	MDI	MDIP	MDIV	Var
bench_65_01	0.2	0.5	0.4	0.3	1.2	**1.5**	0.5	0.3
bench_65_03	0.4	0.8	0.5	0.3	**1.6**	1.5	0.6	0.4
bench_70_01	0.0	0.3	0.3	0.0	0.3	**0.9**	0.2	0.1
bench_70_03	0.1	0.2	0.2	0.1	0.6	**0.9**	0.1	0.2
bench_70_04	0.2	0.6	0.6	0.2	1.0	**1.6**	0.3	0.3
bench_75_01	0.1	**0.5**	0.2	0.3	0.4	0.4	0.2	0.2
bench_75_02	0.4	0.5	0.6	0.5	**1.2**	1.1	0.6	0.2
bench_75_03	0.4	1.0	0.7	0.3	0.8	**2.0**	0.6	0.4
bench_75_04	0.2	0.4	0.2	0.2	**1.4**	1.2	0.4	0.1
bench_80_01	0.3	0.7	0.5	0.1	0.2	**0.9**	0.5	0.2
bench_80_02	0.3	0.4	0.4	0.3	1.0	**1.6**	0.7	0.5
bench_80_03	0.3	0.4	0.4	0.2	1.1	**2.0**	0.3	0.3
bench_80_04	0.1	0.2	**0.8**	0.0	0.3	0.4	0.1	0.1
bench_85_01	0.2	0.3	0.3	0.1	**0.7**	0.6	0.6	0.2
bench_85_02	0.4	0.9	0.6	0.5	**1.6**	1.5	0.6	0.4
bench_85_04	0.3	**1.0**	0.3	0.3	0.9	0.3	0.7	0.3
bench_90_01	0.2	1.0	0.4	0.1	**2.5**	1.7	0.7	0.6
bench_90_02	0.1	0.5	0.3	0.1	**1.4**	1.0	0.4	0.2
bench_90_03	0.3	0.3	0.6	0.0	1.2	**1.4**	0.1	0.3
global avg	0.2	0.6	0.4	0.2	1.0	**1.2**	0.4	0.3

Table 3. Average improvements on the Car Sequencing problem

addition of the *proximity* to this heuristic seems to help the combined heuristic to define more structured fragments. The addition of the *variability* information to this heuristic is deteriorating its performances. This is justified by the poor performance of the *variability* heuristic, because of its lack of diversification.

From those experiments, we can also see that the *adaptive impact* (AI) is the third best performing heuristic. We think that this is due to the LNS step limit for the starting solutions. Indeed, after 35 LNS steps, the information of the adaptive impacts of the variables is quite accurate. The addition of the *proximity* to this heuristic is decreasing the performances of the strategy. This may be due to the fact that some structure is already present in the fragments defined by the *adaptive impact*. Indeed, the variables with large adaptive impacts are linked through the objective function. The performances of the *adaptive impact + variability* (AIV) heuristic could be explained by the fact that the variables with large variability values are different from the ones with large adaptive impacts. Since the combination takes the two measures as equally relevant, the fragment consists of variables with mean adaptive impacts and mean variabilities. Those measures seem incompatible.

The mean execution times in seconds for the strategies are provided in Table 4. Those are the averages over each run of the total times taken by the heuristics to compute the fragment, fix the variables and explore the neighborhood. Those times are all similar. The mean execution time for one run of the random neigh-

borhood heuristic is rather high and may appear counter-intuitive. One could think that without information to compute, the random heuristic will be efficient. However, the computation times for one run depend on the shape of the neighborhood. The neighborhoods defined by our heuristics are more structured, allowing a fast exploration because of the propagation they imply. This fast exploration counterbalances the time spent to compute the information needed by our heuristics.

Heuristic	rand	AI	AIP	AIV	MDI	MDIP	MDIV	Var
Mean time	2.0	2.0	2.3	2.0	2.3	2.1	1.9	2.0

Table 4. Average execution times in seconds on the Car Sequencing problem

4 Conclusion

This work presented various adaptive heuristics to choose the fragments of variables to be relaxed in Large Neighborhood Search. The *mean dynamic impact*, *min/max dynamic impact* and *adaptive impact* measures are trying to approximate the impact of a variable on the value of the objective function. The *variability* measure is linked with the probability of a variable to vary in an improving solution. The last measure - the *proximity* - estimates the link that exist between variables. The heuristics based on these measures are grouped into two categories : *simple* heuristics which use a single measure to choose the fragments and *combined* heuristics that combine two measures to perform their task. The simple category counts four heuristics: the *mean dynamic impact*, the *min/max dynamic impact*, the *adaptive impact* and the *variability* heuristics. There are six combined heuristics that are obtained by combining one of the three impact measures with either the *variability* or the *proximity*.

All these heuristics have been experimented on a new relaxation of the Car Sequencing problem. The results were obtained by the application of a *one step LNS* comparison from particular starting solutions. The measured total number of improvements and their quality showed that all our heuristics (except the *adaptive impact variability*) outperform the standard random heuristic. A comparison between our heuristics reveals that the *mean dynamic impact + proximity* and the *min/max dynamic impact + proximity* heuristics lead to the same fragment choices and are significantly better than the others. These results were explained by the combination of an optimistic measure of the importance of the variables (the *mean dynamic impact – min/max dynamic impact*) which provides a good diversification with an accurate measure of the links that the constraint network induces between the variables (the *proximity*).

The average times for computing the fragment, fixing the variables and searching the neighborhood are comparable for our heuristics and for the random

heuristic. It seems that the structure of the neighborhoods defined by our heuristics makes their exploration faster, reducing thus the overhead of the fragment selection.

Future work includes the testing of our heuristics on other problems to assess their performances in general. It also includes an extended comparison of our heuristics with state of the art domain dependent and independent LNS strategies as well as other incomplete approaches. Those comparisons should be based on time limited runs. Finally, we would like to compare our Car Sequencing relaxation with the one proposed in [1].

Acknowledgment We thank the reviewers for their constructive comments. 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 (National Fund for Scientific Research).

References

1. Laurent Perron, Paul Shaw and Vincent Furnon. Propagation Guided Large Neighborhood Search. CP 2004, LNCS 3258 (2004) 468–481.
2. Nicolas Levasseur, Patrice Boizumault and Samir Loudni. Boosting VNS with Neighborhood Heuristics for Solving Constraint Optimization Problems. LNCS 5296 (2008) 131–145
3. Daniel Godard, Philippe Laborie and Wim Nuijten. Randomized Large Neighborhood Search for Cumulative Scheduling. AAAI 2005.
4. Guy Desaulniers, Eric Prescott-Gagnon and Louis-Martin Rousseau. A Large Neighborhood Search Algorithm for the Vehicle Routing Problem with Time Windows. MIC 2007.
5. 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.
6. Paul Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems CP 1998, LNCS 1520 (1998) 417–431.
7. T. Petit, J.-C. Régin, and C. Bessière. Specific Filtering Algorithms for Over-Constrained Problems. CP 2001, LNCS 2239 (2001) 451–463.
8. Philippe Laborie and Daniel Godard. Self-Adapting Large Neighborhood Search: Application to single-mode scheduling problems. Proceedings MISTA-07, Paris, 2007, 276-284.