Variable Objective Large Neighborhood Search: A practical approach to solve over-constrained problems

Pierre Sschaus
UCLouvain, ICTEAM
Place sainte barbe 2,
1348 Louvain-la-Neuve, Belgium
pierre.schaus@uclouvain.be

***Abstract*—Everyone having used Constraint Programming (CP) to solve hard combinatorial optimization problems with a standard exhaustive Branch & Bound Depth First Search (B&B DFS) has probably experienced scalability issues. In the 2011 Panel of the Future of CP, one of the identified challenges was the need to handle large-scale problems. In this paper, we address the scalability issues of CP when minimizing a sum objective function. We suggest extending the Large Neighborhood Search (LNS) framework enabling it with the possibility of changing dynamically the objective function along the restarts. The motivation for this extended framework - called the Variable Objective Large Neighborhood Search (VO-LNS) - is solving efficiently a real-life over-constrained timetabling application. Our experiments show that this simple approach has two main benefits on solving this problem: 1) a better pruning, boosting the speed of LNS to reach high quality solutions, 2) a better control to balance or weight the terms composing the sum objective function, especially in over-constrained problems.**

***Keywords*-constraint programming; large neighborhood search; over-constrained problems; sum objective;**

I. Introduction and Motivation

One of the main weaknesses of CP is its inability to deal efficiently with (weighted) sums often introduced by the modeler as the objective function to minimize. As noted in [20], a lot of optimization problems present this aspect of minimizing (or maximizing) the (weighted) sum of some sub-objective variables o_i:

$$z = \sum_i o_i$$

It is well known that sum constraints in CP generally present a poor filtering in terms of lower bounds compared to equivalent Integer Programming (IP) models. Most of the CP solvers achieve bound-consistency on sum constraints without any possible communication with other constraints other than through the domain store (current domain for each variable). It means that the lower bound of the objective variable is computed from the sum constraint by considering minimum values of the sub-objective variables: $\underline{z} = \sum_i \underline{o_i}$. It is often a frustration for people coming from the IP community to discover the weakness of CP on this aspect since they are used to model with sum objective function without necessarily affecting the efficiency of their model. The difference is that IP relies on a linear programming (LP) relaxation solved by the simplex algorithm (see [30] for introduction to IP). In the simplex all the linear constraints and the linear objective are considered together providing a generally tighter lower bound than the one offered by CP. The next example illustrates the weak filtering of CP on sum objective functions.

Example 1.1: (Inspired from [20]) $z = o_1 + o_2$ has to be minimized while satisfying the constraints $o_1 = 3x + 2y$ and $o_2 = w - 2x - y$ with $Dom(x) = Dom(y) = [0, 5]$ and $Dom(w) = [5, 10]$. The CP filtering will lead to sub-objective lower bounds $\underline{o_1} = 3 \cdot \underline{x} + 2 \cdot \underline{y} = 0$ and $\underline{o_2} = \underline{w} - 2 \cdot \overline{x} - \overline{y} = -10$. The objective lower bound will be $\underline{z} = \underline{o_1} + \underline{o_2} = -10$. The purpose of this example is to illustrate that the communication between the constraints only happens through the domain store. On this example, an LP solver using the simplex algorithm discovers the lower bound 5 for z.

As mentioned by van Hoeve in his tutorial [26], many (most) industrial problems are essentially over-constrained. Solving efficiently a real over-constrained timetabling application introduced in Section III-B is the main motivation for this work. It is common in those problems to minimize a sum objective function of the violations.

Over-Constrained Problems: The meta-constraint framework introduced in [16] associates a cost variable to each soft constraint to represent its violation. Many soft global constraints and their filtering algorithms have been developed over the last decade: the soft all-different [17], soft global cardinality [28] soft-cumulative [3], etc (see [27] for more of them). van Hoeve draws this important conclusion in his chapter dedicated to over-constrained problems [27]:

Many research challenges remain in this area. Perhaps the most important one is the issue of aggregating effectively different soft global constraints. It is likely that a weighted sum of the associated cost variables is not the most effective aggregation. Other approaches, such as minimizing the maximum over all cost variables, or applying a (soft) balancing constraint to the cost variables, appear to be more

promising. Although not the most effective one in terms of filtering, minimizing the sum of violations is probably the most intuitive one and is also the one proposed in [19] and [11] to solve real over-constrained nurse rostering problems.

The consequence of a weak filtering on lower bounds with the introduction of sum objective constraints is that a B&B DFS exploration can quickly get stuck in a small region of a huge search tree. The early decisions taken in the DFS exploration have little chance to be reconsidered and only a small portion of the search space can be explored to discover good solutions. The answer of Shaw to this problem was the introduction of the Large Neighborhood Search (LNS) framework [24].

LNS: The idea of this incomplete method is to combine the expressiveness of CP and the efficiency of Local Search (LS): CP is used as the slave technique to explore a (large) neighborhood around the current best solution in order to improve it. The neighborhood exploration consists in exploring a smaller very constrained problem keeping fixed part of the structure from the current best solution. LNS has been successfully used to solve large-scale complex problems such as vehicle routing [1], [24], scheduling [15], [7] and assignment/bin-packing problems [22], [10].

Contribution: Our contribution is to extend the LNS framework to improve the filtering when the objective function to optimize is a (weighted) sum, for instance resulting from the aggregation of the violations of soft global constraints. We propose to optimize a few terms of the sum at a time; changing these terms dynamically along the LNS restarts (standard LNS with a variable objective function). We call this extended framework the Variable Objective LNS (VO-LNS). The benefits of VO-LNS are

- a stronger pruning during the branch and bound;
- a full control on the prioritization among the different sub-objectives.

This work does not attempt to improve the poor communication of sum constraints with other constraints in the constraint store as was proposed in [20]. Instead it minimizes the effect by reducing the number of free terms in the sum during the branch and bound search.

Related Work: The idea of changing dynamically the objective function along iterations is not new and has been successfully applied in various contexts. In continuous optimization, the metric and the search direction is changed dynamically in [2], [4]. The guided local search (GLS) meta-heuristic [29] uses an objective function augmented by penalties changing dynamically. The Variable Neighborhood Search meta-heuristic (VNS) changes the neighborhood during search [12], [5]. The motivation of GLS (VNS) is the recognition that a local optimum with respect to one objective function (neighborhood) may not be locally optimal for another objective function (neighborhood). The Adaptive Large Neighborhood Search (ALNS) heuristic proposed in [21], [18] extends the LNS heuristic by allowing multiple relaxation and repair methods to be used within the same search using weights adjusted dynamically as the search progresses. The common motivation of GLS, VNS and ALNS meta-heuristics is to escape from local optima. As for GLS, we believe that the VO-LNS can also help to escape from local minima but the main goal of VO-LNS is to improve the filtering in the context of sum objective functions to discover better solutions in less time.

Outline: Section II formally recalls the classical LNS in CP and its parameters. It then introduces the VO-LNS framework. In Section III the VO-LNS framework is tested on two different over-constrained problems illustrating the effectiveness of VO-LNS in terms of time and solution quality. The first problem is an artificial rostering problem while the second one is a real-life timetabling problem occurring in a hospitality school. Section IV discusses the limitations of VO-LNS. Section V offers some perspectives and future work we plan to explore with VO-LNS. We conclude in Section VI.

II. Variable Objective LNS

We first recall what LNS is and its parameters. We then introduce the extended VO-LNS framework.

The initial constrained optimization model has the following form minimizing a variable z resulting from a sum constraint:

$$\begin{array}{ll} \text{Minimize} & z = o_1 + o_2 + \ldots + o_n \\ \text{Subject to} & constraints \end{array} \quad (1)$$

To turn the original CP model into a LNS, the user needs to provide:

- A relaxation procedure. This procedure (also called fragment selection) is responsible for defining the neighborhood. It adds some constraints to the problem coming from the structure of the current best solution[1] while allowing some flexibility for re-optimization.
- A search limit. This limit although optional is responsible to trigger a new restart when it is reached to avoid spending too much time in the neighborhood exploration. It can for instance be a time limit, or a limit on the number of backtracks.

In the extended VO-LNS framework one would state an optimization model as follows:

$$\begin{array}{ll} \text{Optimize} & obj = (obj_1, obj_2, \ldots, obj_m) \\ \text{Subject to} & constraints \end{array} \quad (2)$$

The only difference is that several objectives can be stated rather than only one. Each sub-objective can either be a minimization or maximization encapsulating one variable and can be set independently into three different filtering modes during the B&B search:

[1]It is quite easy in any modern CP solver to record the current best solution during the search, for instance through a call-back mechanism.

1) *No-Filtering*: it means that it is deactivated, having no impact at all.
2) *Weak-Filtering*: when a solution is found during the B&B DFS, the bound of the objective is updated such that the next found solution is better or equal to the bound with respect to this objective.
3) *Strong-Filtering*: when a solution is found during the B&B DFS, the bound of the objective is updated such that the next found solution is strictly improving the bound of this objective.

Example 2.1: Let us consider two variables $x_1, x_2 \in [0..2]$ and the minimization of $2x_1 + x_2$. We assume the B&B DFS first assigns variable x_1 then x_2 in decreasing value order. Three minimization objectives are introduced $obj_1 = 2x_1$, $obj_2 = x_2$ and $obj_3 = 2x_1 + x_2$. We consider three different configurations for the filtering mode of these objectives. For each configuration, the solutions discovered during the B&B DFS are reported in Table I from the first to the last (optimal) solution. N, W, S denote respectively *No-Filtering*, *Weak-Filtering* and *Strong-Filtering* modes. As can be seen, some solutions are not considered when the filtering mode is stronger. The setting W,W,S was able to find the same optimal solution with respect to to obj_3 as in setting N,N,S. But the best solution obtained with setting S,W,S is sub-optimal with respect to obj_3 (2 instead of 0).

		$\mathbf{obj_1}$	$\mathbf{obj_2}$	$\mathbf{obj_3}$
$\mathbf{x_1}$	$\mathbf{x_2}$	**N**	**N**	**S**
2	2	4	2	6
2	1	4	1	5
2	0	4	0	4
1	1	2	1	3
1	0	2	0	2
0	1	0	1	1
0	0	0	0	0
		W	**W**	**S**
2	2	4	2	6
2	1	4	1	5
2	0	4	0	4
1	0	2	0	2
0	0	0	0	0
		S	**W**	**S**
2	2	4	2	6
1	2	2	2	4
0	2	0	2	2

Table I: Impact of the filtering mode configurations on the solutions discovered during the B&B DFS.

Note that model (2) is more expressive than model (1). We can easily state the initial model (1) into an equivalent VO-LNS formulation with just one sub-objective being the minimization of the summation of the terms:

$$\begin{aligned} &\text{Optimize} \quad obj = (obj_1) \\ &\text{Subject to} \quad constraints \end{aligned} \tag{3}$$

with $obj_1 = \text{minimize}(z = o_1 + \ldots + o_n)$ set into the *Strong-Filtering* mode. In VO-LNS we are interested in dynamically changing the filtering mode of sub-objectives along the restarts. Our initial model (1) is expressed as follows:

$$\begin{aligned} &\text{Optimize} \quad obj = (obj_1, obj_2, ..., obj_n, obj_{n+1}) \\ &\text{Subject to} \quad constraints \end{aligned} \tag{4}$$

with $obj_i = \text{minimize}(o_i)$ for all $i \in [1..n]$ and $obj_{n+1} = \text{minimize}(z = o_1 + \ldots + o_n)$. Note that we add the original sum objective as one of the sub-objective. This sub-objective obj_{n+1} is set into the *Strong-Filtering* mode such that the filtering of the B&B DFS is at least as strong as in the original model.

The next example illustrates why having more than one variable set into *Strong/Weak Filtering* mode can trigger the filtering of the constraints earlier during the B&B DFS, helping to reduce the search space.

Example 2.2: Assume the minimization of $o_1 + o_2$ with variables $o_1, o_2 \in [0..2]$ involved in two optimization constraints[2] $C_1(vars_1, o_1), C_2(vars_2, o_2)$. Assume further that C_1 (C_2) is only able to filter some values from $vars_1$ ($vars_2$) when the maximum value of the domain of o_1 (o_2) is 1. Three objectives to minimize are introduced: $obj_1 = o_1$, $obj_2 = o_2$ and $obj_3 = o_1 + o_2$, each objective maintains its own upper bound. We consider two different configurations for the filtering mode of these objectives. The first one N, N, S is equivalent to standard minimization of $o_1 + o_2$ since the other objectives are deactivated. The second one S, W, S is a possible setting in the VO-LNS framework. For both configurations, we report on Table II the worst case scenario with respect to the decreasing of the upper bounds of the objectives along the B&B DFS (we assume 0 lower bounds). For the first setting, some filtering is triggered only when the upper bound on $o_1 + o_2$ reaches 1 since the sum constraint can set neither the maximum of o_1 nor o_2 to 1 when the upper bound on $o_1 + o_2$ is greater than 1. In the second setting, the filtering on C_1 already happens after the first solution has been discovered during the B&B DFS since obj_1 upper bound is immediately set to 1.

$\mathbf{obj_1}$	$\mathbf{obj_2}$	$\mathbf{obj_3}$	$\mathbf{C_1}$	$\mathbf{C_2}$
N	**N**	**S**	filtering?	
2	2	4	No	No
2	2	3	No	No
2	2	2	No	No
1	1	1	Yes	Yes
S	**W**	**S**	filtering?	
2	2	4	No	No
1	2	3	Yes	No

Table II: Impact of the filtering mode configurations on the objective upper bounds update and the filtering during the search.

With this set of sub-objectives in problem formulation (4), the relaxation procedure of VO-LNS has one more

[2]for instance $softgcc$ constraints [28], [23].

responsibility, which is changing dynamically the filtering mode of the sub-objectives for the next restart. Here is for instance one strategy we experimented and found successful on over-constrained problems:

- Never change obj_{n+1} *i.e.* keep it its *Strong-Filtering* filtering mode.
- Select the worst objective $obj_k \in \{obj_1, \ldots, obj_n\}$ in the current best solution (the one with maximum value in case of a minimization). Set obj_k into *Strong-Filtering* mode. Set the other ones either into *Weak-Filtering* or *No-Filtering* mode.
- Apply a relaxation procedure dependent from the selected objective obj_k to give it the largest chance of success to improve during next restart.

Selecting preferably the worst objectives at each restart guides the search toward balanced solutions with respect to each of the terms, which is generally a desirable property when solving over-constrained problems.

III. Experiments

We propose to solve two over-constrained problems to compare the performances of VO-LNS to classical LNS. The first problem tested in Section III-A is an artificial rostering problem that anyone interested can easily reproduce without much implementation effort. The second problem experimented in Section III-B is a real timetabling application that motivated the development of VO-LNS. All experiments were conducted with the OscaR open-source solver [14].

A. Artificial Rostering Problem

This problem is an artificial over-constrained rostering problem but presenting a similar structure to real ones. A 15×15 matrix of variables is defined with vertical and horizontal cardinality constraints. The domain of each variable is a set of values obtained by sampling randomly 5 times numbers in $[1..15]$. One global cardinality constraint is added on each line and each column requiring that every value $v \in [1..15]$ should appear exactly once. The way the domains are generated do not guarantee that the cardinality requirements are all feasible together. This problem might be over-constrained. Each cardinality constraint is relaxed into the soft equivalent [28] introducing 30 violation variables[3]. We denote z_i ($i \in [1..30]$) the violation variable of rows and columns and $z_{tot} = \sum_{i \in [1..30]} z_i$, the total violation of the problem. The objective is to minimize z_{tot} and ideally the violations z_i's should be balanced as much as possible. Three different settings (relaxation procedures and objective configurations) are experimented.

- A: Randomly relax 10% of the variables. The only objective is the minimization of the variable z_{tot} set into the *Strong-Filtering* filtering mode
- B: Select the column or row with the largest violation in the current best solution and relax the variables of this row or column. Also relax randomly 10% of the variables as in A. The selected row or column becomes tabu during $2 + k$ restarts where k is a random number taken on $[0..2]$ to ensure diversification. This tabu procedure is used to avoid over-selecting the same row or column along restarts. As in A, the only objective is the minimization of the variable z_{tot} set into the *Strong-Filtering* mode.
- C: The relaxation is the same as in B. All the minimization objectives are added to the model $obj = (obj_1, \ldots, obj_{30}, obj_{tot})$ with $obj_i = \text{minimize}(z_i)$ and $obj_{tot} = \text{minimize}(z_{tot})$. Objective obj_{tot} is set into the *Strong-Filtering* filtering mode and remains in that mode. The objective corresponding to the relaxed row or column is set into the *Strong-Filtering* mode while others (except obj_{tot}) are set into the *Weak-Filtering* mode.

Note that settings A and B are both classical LNS (only the relaxation procedure differs) while setting C uses the VO-LNS framework since the filtering mode of objectives is changed dynamically during restarts.

Figure 1 depicts the evolution of z_{tot} along 400 restarts with a 50 backtrack limit using settings A, B and C.

Let us first compare A and B who only differ in the relaxation procedure. The more sophisticated relaxation used in B allows a faster decrease of the objective function during the first 50 restarts. Then A is decreasing faster than B and reaches a better objective value at the end of the 400 restarts.

C is using the same relaxation as B but differs in the objective functions. We can see that C is the clear winner over the two other methods decreasing much faster and reaching the best objective function at the end. This is because the filtering of the objectives is much stronger and aggressive in C: at each restart two sub-objectives are set into *Strong-Filtering* and the other ones are set into *Weak-Filtering* mode.

Since we are also interested into the distribution of the violations we propose to analyze the standard deviation of the violations $z_1, \ldots, z_{30}$. Ten instances were generated and solved using settings A, B and C. The results are given on Table III. Surprisingly the idea of using an LNS relaxation focusing on a sub-objective (setting B) obtains worse results than the random relaxation with standard LNS (setting A). However, VO-LNS (setting C) strongly reinforces this strategy by offering a more aggressive filtering of the sub-objective functions. One can see that C is consistently the best approach to minimize the total violation z_{tot}. It also seems to be the best approach to obtain balanced violations. It fails to have the smallest standard deviation only in two instances.

[3]We use the *value based* violation [28] and the filtering algorithm introduced in [23]

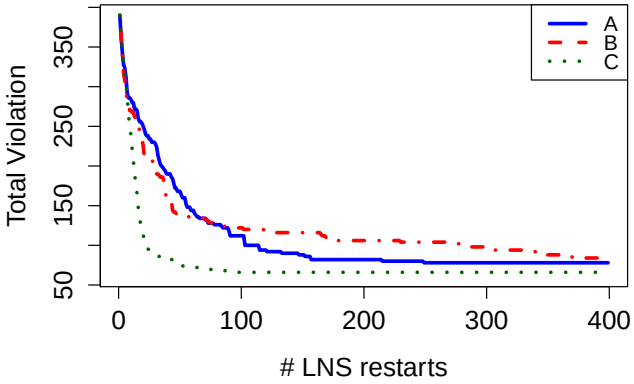

Figure 1: Evolution of the total violation along restarts obtained on an artificial rostering instance with LNS settings A, B and C

z_{tot}			standard deviation		
A	B	C	A	B	C
78	84	**66**	**1,30**	1,71	1,42
70	104	**60**	**1,18**	1,48	1,29
68	74	**60**	1,72	1,63	**1,05**
80	100	**66**	1,69	1,69	**1,52**
78	100	**56**	1,75	1,69	**1,17**
68	108	**54**	1,14	2,13	**0,96**
74	106	**66**	1,46	1,87	**1,21**
74	108	**62**	1,72	1,85	**1,23**
70	114	**60**	1,18	2,12	**1,05**
82	126	**66**	1,44	2,12	**1,32**

Table III: Total violation and standard deviation obtained on 10 artificial rostering instances with LNS settings A, B and C

B. Real-life over-constrained timetabling problems

This timetabling problem comes from a hospitality school planning the course activities for its students for the next semester. Solving efficiently this over-constrained problem was the original motivation for this work.

Students are divided into n groups. Each group must be scheduled two activities every week (one morning and one afternoon activity) during m weeks[4]. There are thus $n \times m \times 2$ decision variables that must be assigned an activity. There are $p+1$ different activities with activity p representing the empty slot (in our instance $n = 15$, $m = 20$ and $p = 16$). A table view of the decision variables $\mathbf{x}$ is given in Table IV. $x_{i,j}^{AM} \in [0..p]$ ($x_{i,j}^{PM} \in [0..p]$) is the morning (afternoon) activity assigned to group i in week j.

Before describing the constraints, we introduce some notations to extract *slices* and *rows* of variables out of table $\mathbf{x}$.

[4]A group receives the same morning (afternoon) activity all the days of the week.

$Week_1$			$Week_j$			$Week_m$	
$x_{1,1}^{AM}$	$x_{1,1}^{PM}$	...	$x_{1,j}^{AM}$	$x_{1,j}^{PM}$	...	$x_{1,m}^{AM}$	$x_{1,m}^{PM}$
...	...	...	...	...	...	...	...
$x_{i,1}^{AM}$	$x_{i,1}^{PM}$	...	$x_{i,j}^{AM}$	$x_{i,j}^{PM}$	...	$x_{i,m}^{AM}$	$x_{i,m}^{PM}$
...	...	...	...	...	...	...	...
$x_{n,1}^{AM}$	$x_{n,1}^{PM}$	...	$x_{n,j}^{AM}$	$x_{n,j}^{PM}$	...	$x_{n,m}^{AM}$	$x_{n,m}^{PM}$

Table IV: Table $\mathbf{x}$ of the decision variables of the hospitality school timetabling problem.

Notations: We represent by $\mathbf{x}_{i,*}^{AM}$ the vector of AM variables related to group i: $(x_{i,1}^{AM}, x_{i,2}^{AM}, \ldots, x_{i,m}^{AM})$. Similarly $\mathbf{x}_{i,*}^{PM}$ is the vector of PM variables related to group i and $\mathbf{x}_{i,*}^{*}$ the vector of AM and PM variables related to group i: $(x_{i,1}^{AM}, x_{i,1}^{PM}, x_{i,2}^{AM}, x_{i,2}^{PM}, \ldots, x_{i,m}^{AM}, x_{i,m}^{PM})$. We also define $\mathbf{x}_{*,j}^{AM}$ the vector of AM variables related to week j: $(x_{1,j}^{AM}, x_{2,j}^{AM}, \ldots, x_{n,j}^{AM})$. Vectors $\mathbf{x}_{*,j}^{PM}$ and $\mathbf{x}_{*,j}^{*}$ are defined similarly.

Constraints: The *horizontal* constraints (specific to a group) are:

- Each activity can be scheduled a minimum and maximum number of times over the schedule of a group. This constraint is modeled using one global cardinality constraint (GCC) over each vector $\mathbf{x}_{i,*}^{*}$.
- Some activities if scheduled, must be scheduled both in the morning and in the afternoon of the week. These constraints are handled using table constraints.
- Some activities if scheduled must be scheduled in the morning and in the afternoon during two consecutive weeks. These constraints are also handled using table constraints every two weeks.
- Some theoretical courses are pre-assigned.

The *vertical* constraints related to a week are:

- On a given week, each activity can appear a minimum and maximum number of times over the morning slots of the n groups. This constraint is modeled with one GCC over each vector $\mathbf{x}_{*,j}^{AM}$. We have the same kind of constraint over the afternoon slots $\mathbf{x}_{*,j}^{PM}$. We denote the constraint on column j AM (PM) slots by gcc_j^{AM} (gcc_j^{PM}).
- Some activities must be scheduled every week in at least one group. This constraint is handled with one GCC on variables $\mathbf{x}_{*,j}^{*}$ every week j, with minimum occurrences set to 1 on activities that must be scheduled at least once. We denote the constraint on column j by gcc_j^{Week}.

The problem instance[5] that needs to be solved appears to be over-constrained. Every vertical cardinality constraint gcc_j^{AM}, gcc_j^{PM}, gcc_j^{Week}, $\forall j \in [1..m]$ is turned into a soft

[5]Available upon request to the author.

cardinality constraint and their respective violation variables are $z_j^{AM}, z_j^{PM}, z_j^{\text{Week}}$.

The VO-LNS setting is similar to setting C used on the previous artificial rostering problem. One minimization objective in introduced for each violation variable obj_j^{AM}, obj_j^{PM}, obj_j^{Week}. One objective $obj_{tot} = \text{minimize}(z_{tot} = \sum_j (z_j^{AM} + z_j^{PM} + z_j^{\text{Week}}))$ is also added. Objective obj_{tot} is set into the *Strong-Filtering* filtering mode and remains in that mode. Others (except obj_{tot}) are set initially in the *No-Filtering* mode.

We designed a LNS relaxation procedure to favor the chance of improvement along restarts on this problem. On even restart numbers, two weeks are randomly relaxed. On odd restart numbers two groups are randomly relaxed. On every restart, the objective with the worst violation is selected. The variables appearing in the week corresponding to selected objective variable are also relaxed. The selected objective is set tabu for a random number of iterations between 3 and 5. We used 1000 LNS restarts in our experiments; each with a limit of 100 backtracks.

Figure 2 depicts the evolution of the total violation z_{tot} with standard LNS and with VO-LNS also setting the selected nontabu objective into *Strong-Filtering* mode while others (except obj_{tot}) are set into *Weak-Filtering* mode. Table V reports the final violations obtained at the end of the 1000 restarts and the time required to reach it.

As can be seen on Figure 2, the total violation decreases much faster along the iterations with VO-LNS compared to standard LNS. Also the final objective is better and the different morning and afternoon violations are more balanced. Interestingly the time required to achieve the 1000 restarts is much shorter for VO-LNS (44 vs. 155 seconds). This is because most of the restarts with standard LNS were caused by the limit of 100 backtracks while for VO-LNS, the search tree exploration was exhausted in 894/1000 restarts. This demonstrates the stronger filtering obtained with VO-LNS over standard LNS. Based on these results, we believe that VO-LNS can offer an improved any-time behavior over classical LNS on this type of problems.

	Tot	AM	PM	Week	N	time(s)
LNS	93	53	38	2	18	155
VO-LNS	82	40	39	3	894	44

Table V: Some indicators comparing LNS and VO-LNS on the real-life timetabling problem. Columns represent namely the final total violation $\sum_j (z_j^{AM} + z_j^{PM} + z_j^{\text{Week}})$, total violation on AM columns $\sum_j z_j^{AM}$, total violation on PM columns $\sum_j z_j^{PM}$, total week violation $\sum_j z_j^{\text{Week}}$, the number of exhausted LNS restart and the total time in seconds.

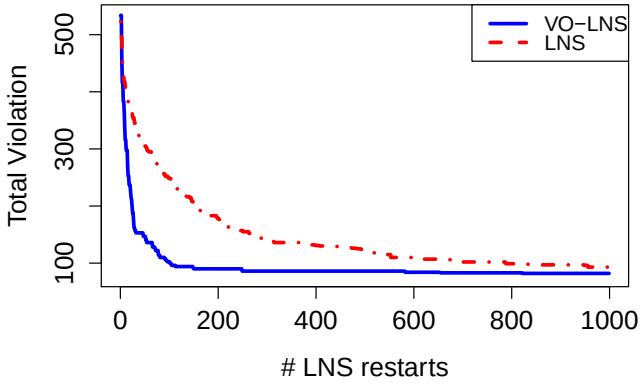

Figure 2: Evolution of the total violation along restarts obtained on the real-life timetabling problem with LNS and with VO-LNS.

We have tested the approach on 5 more instances with a timeout of 300 seconds[6] for each run. Table VI gives the minimum, average, and maximum violation over 10 runs for each instance. As can be seen, only for one instance (instance 4), it is not clear whether the VO-LNS allows to improve the violation or not. For the other four instances, VO-LNS outperforms clearly standard LNS.

VO-LNS			LNS		
min	avg	max	min	avg	max
99	**99,5**	**100**	103	105	107
122	**127,6**	**132**	121	137,8	149
116	**116,7**	**119**	156	166,8	185
136	144,7	159	140	**143,1**	**147**
82	**85,6**	**94**	94	101,4	112

Table VI: Minimum, average, and maximum violation over 10 runs for each instance.

IV. Limitations of VO-LNS

The VO-LNS approach is not suited for every problem. In particular when the tension between the sub objectives is too important, the VO-LNS may fail to help and even worse, it could prevent the LNS to escape from local optima. One such example is the large-scale power restoration problem [25] combining routing and power outage objectives. On this kind of problem with high tension between the sub objectives, a multi-stage decomposition approach seems more appropriate.

However, there are problems where the tension between the sub-objectives is not that strong. On over-constrained

[6]This timeout was chosen such that the objective does not improve anymore for both LNS and VO-LNS.

problems (such as the one treated in Section III-B), minimizing the violation of one constraint may only have limited, local, impact. On this type of problems, the VO-LNS approach can be effective as demonstrated in the experimental section.

The idea of focusing LNS relaxations to sub-objectives is the key idea that actually underlies most LNS neighborhoods (such as the one used in the real-life over constrained problem of Section III-B). Our experiments have demonstrated that the VO-LNS can reinforce this strategy by offering a more aggressive filtering of the sub-objective functions. To some extend, VO-LNS can be seen as part of the definition of the neighborhood helping it to focus on different selected sub-objectives at each restart.

V. Future work and perspectives

Some perspectives and future work we plan to do to extend the VO-LNS framework are given below.

Objective selection strategies: We have experimented a tabu meta-heuristic on top of the VO-LNS for the objective selection. Many more meta-heuristics could be developed such as Machine Learning (ML) based approaches to discover combinations of objectives that should be set together into Strong/Weak tightening mode. ML approaches on top of LNS have already been used in [7], [8], [9]. We also plan to study the impact of the order in which the sub-objectives are considered. We would like to imagine general strategy for VO-LNS.

Over-constrained optimization problems: We believe that VO-LNS is particularly well suited for this kind of problems. The VO-LNS could first focus on the feasibility aspect of the problem by minimizing the violations before focusing on the optimization without degrading the violations objectives.

Earliness, Tardiness problems: VO-LNS could be used to minimize violations in just-in-time scheduling problems. Those problems are hard to solve with CP because of the weak propagation when aggregating all the earliness/tardiness costs. Focusing at each restart on the objective tardiness of a restricted number of activities may improve the filtering. Combining it with better propagation as in [6] and [13] may give interesting results.

VI. Conclusion

This work introduced the VO-LNS framework allowing the optimization of several objectives at once. At each restart the filtering behavior of the different objectives can be changed between $\{$ *Weak,Strong,No* $\}$*-Filtering*. This simple and pragmatic method seems particularly well suited to optimize over-constrained problems where we generally minimize the sum of the violations. Our experiments showed that the filtering improvement of the objective introduced with VO-LNS allowed to reach high quality solutions in a much smaller number of restarts and less time than classical LNS. VO-LNS also offers a better control to balance the violation of the importance of the sub-objectives to optimize. One of the main advantage of VO-LNS is that it doesn't require changing much the CP solvers already offering standard LNS.

References

[1] R. Bent and P.V. Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, 33(4):875–893, 2006.

[2] W.C. Davidon. *Variable metric method for minimization*. Argonne National Laboratory, 1959.

[3] A. De Clercq, T. Petit, N. Beldiceanu, and N. Jussien. A soft constraint for cumulative problems with over-loads of resource. In *Principles and Practice of Constraint Programming*, 2011.

[4] R. Fletcher and M.J.D. Powell. A rapidly convergent descent method for minimization. *The Computer Journal*, 6(2):163–168, 1963.

[5] P. Hansen, N. Mladenović, and J.A. Moreno Pérez. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.

[6] András Kovács and J Christopher Beck. A global constraint for total weighted completion time for unary resources. *Constraints*, 16(1):100–123, 2011.

[7] P. Laborie and D. Godard. Self-adapting large neighborhood search: Application to single-mode scheduling problems. *Proceedings MISTA-07, Paris*, pages 276–284, 2007.

[8] Jean-Baptiste Mairy, Yves Deville, and Pascal Van Hentenryck. Reinforced adaptive large neighborhood search. In *8th Workshop on Local Search techniques in Constraint Satisfaction (LSCS 2011). A Satellite Workshop of CP*, Perugia, Italy, 2011.

[9] Jean-Baptiste Mairy, Pierre Schaus, and Yves Deville. Generic adaptive heuristics for large neighborhood search. In *Seventh International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS2010). A Satellite Workshop of CP*, 2010.

[10] D. Mehta, B. OSullivan, and H. Simonis. Comparing solution methods for the machine reassignment problem. In *Principles and Practice of Constraint Programming*, pages 782–797. Springer, 2012.

[11] J.P. Métivier, P. Boizumault, and S. Loudni. Solving nurse rostering problems using soft global constraints. *Principles and Practice of Constraint Programming*, pages 73–87, 2009.

[12] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[13] Jean-Noël Monette, Yves Deville, and Pascal Van Hentenryck. Just-in-time scheduling with constraint programming. In *The 19th International Conference on Automated Planning and Scheduling*, Tessaloniki, Greece, 19/09/2009 2009.

[14] OscaR Team. OscaR: Scala in OR, 2012. Available from https://bitbucket.org/oscarlib/oscar.

[15] D. Pacino and P. Van Hentenryck. Large neighborhood search and adaptive randomized decompositions for flexible jobshop scheduling. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, pages 1997–2002. AAAI Press, 2011.

[16] T. Petit, J.C. Régin, and C. Bessière. Meta-constraints on violations for over constrained problems. In *Tools with Artificial Intelligence, 2000. ICTAI 2000. Proceedings. 12th IEEE International Conference on*, pages 358–365. IEEE, 2000.

[17] T. Petit, J.C. Régin, and C. Bessière. Specific filtering algorithms for over-constrained problems. In *Principles and Practice of Constraint Programming*, pages 451–463. Springer, 2001.

[18] D. Pisinger and S. Ropke. Large neighborhood search. *Handbook of metaheuristics*, pages 399–419, 2010.

[19] J.C. Régin. Global constraints and filtering algorithms. *Constraint and Integer Programming–Towards a Unified Methodology*, pages 89–129, 2003.

[20] J.C. Régin and T. Petit. The objective sum constraint. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 190–195, 2011.

[21] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[22] 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*, 16(2):125–147, 2011.

[23] P. Schaus, P. Van Hentenryck, and A. Zanarini. Revisiting the soft global cardinality constraint. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 307–312, 2010.

[24] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. *Principles and Practice of Constraint Programming*, pages 417–431, 1998.

[25] Ben Simon, Carleton Coffrin, and Pascal Van Hentenryck. *Randomized adaptive vehicle decomposition for large-scale power restoration*. Springer, 2012.

[26] W.J. van Hoeve. Soft global constraints, tutorial. In *tutorial given at Principles and Practice of Constraint Programming*, 2009.

[27] W.J. van Hoeve. *Over-Constrained Problems. Chapter from Hybrid Optimization: The Ten Years of CPAIOR*, volume 45. Springer, 2010.

[28] W.J. van Hoeve, G. Pesant, and L.M. Rousseau. On global warming: Flow-based soft global constraints. *Journal of Heuristics*, 12(4):347–373, 2006.

[29] Voudouris. Guided local search for combinatorial optimisation problems. *PhD Thesis*, 1997.

[30] L.A. Wolsey. *Integer Programming*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1998.