ECONSTOR
Make Your Publications Visible.

A Service of

ZBW Leibniz-Informationszentrum Wirtschaft
Leibniz Information Centre for Economics

Adamo, Tommaso; Ghiani, Gianpaolo; Guerriero, Emanuela; Manni, Emanuele

Article

Automatic instantiation of a Variable Neighborhood Descent from a Mixed Integer Programming model

Operations Research Perspectives

Provided in Cooperation with:
Elsevier

Suggested Citation: Adamo, Tommaso; Ghiani, Gianpaolo; Guerriero, Emanuela; Manni, Emanuele (2017) : Automatic instantiation of a Variable Neighborhood Descent from a Mixed Integer Programming model, Operations Research Perspectives, ISSN 2214-7160, Elsevier, Amsterdam, Vol. 4, pp. 123-135, https://doi.org/10.1016/j.orp.2017.09.001

This Version is available at:
https://hdl.handle.net/10419/178281

Standard-Nutzungsbedingungen:

Die Dokumente auf EconStor dürfen zu eigenen wissenschaftlichen Zwecken und zum Privatgebrauch gespeichert und kopiert werden.

Sie dürfen die Dokumente nicht für öffentliche oder kommerzielle Zwecke vervielfältigen, öffentlich ausstellen, öffentlich zugänglich machen, vertreiben oder anderweitig nutzen.

Sofern die Verfasser die Dokumente unter Open-Content-Lizenzen (insbesondere CC-Lizenzen) zur Verfügung gestellt haben sollten, gelten abweichend von diesen Nutzungsbedingungen die in der dort genannten Lizenz gewährten Nutzungsrechte.

Terms of use:

Documents in EconStor may be saved and copied for your personal and scholarly purposes.

You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.

If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.

http://creativecommons.org/licenses/by-nc-nd/4.0/

WWW.ECONSTOR.EU

Mitglied der Leibniz-Gemeinschaft

Operations Research Perspectives 4 (2017) 123–135

Contents lists available at ScienceDirect

Operations Research Perspectives

journal homepage: www.elsevier.com/locate/orp

Automatic instantiation of a Variable Neighborhood Descent from a Mixed Integer Programming model

Tommaso Adamo, Gianpaolo Ghiani, Emanuela Guerriero, Emanuele Manni*

Dipartimento di Ingegneria dell'Innovazione, Università del Salento, Via per Monteroni, Lecce 73100, Italy

ARTICLE INFO

Article history:
Received 18 November 2016
Revised 15 September 2017
Accepted 15 September 2017
Available online 19 September 2017

Keywords:
Mixed Integer Programming
Variable Neighborhood Descent
Semantic features

ABSTRACT

In this paper we describe the automatic instantiation of a Variable Neighborhood Descent procedure from a Mixed Integer Programming model. We extend a recent approach in which a single neighborhood structure is automatically designed from a Mixed Integer Programming model using a combination of automatic extraction of semantic features and automatic algorithm configuration. Computational results on four well-known combinatorial optimization problems show improvements over both a previous model-derived Variable Neighborhood Descent procedure and the approach with a single automatically-designed neighborhood structure.

© 2017 The Authors. Published by Elsevier Ltd.
This is an open access article under the CC BY-NC-ND license.
(http://creativecommons.org/licenses/by-nc-nd/4.0/)

1. Introduction

The design of a metaheuristic algorithm comprises a number of steps, including the definition of "good" neighborhood structures in the solution space as well as the "tuning" of a number of parameters characterizing the higher level search strategy. Typically, to make the metaheuristic efficient, both the local and the global improvement mechanisms must be tailored to: (a) the specific problem; (b) the distribution of the instances to be solved (i.e., the *reference instance population*). Other relevant factors are: the time limit for the exploration of a single neighborhood structure, the time limit for the whole procedure as well as the hardware at disposal. The design process may take days, weeks or even months and is typically done by human experts through a number of steps, including:

(a) problem analysis: the problem structure as well as the characteristics of the reference instance population are thoroughly examined in order to extract meaningful properties and features;
(b) literature scouting: the literature related to the same or similar problems is analyzed trying to identify algorithms and approaches that have proven to be successful;
(c) neighborhood structures design: a number of tentative neighborhood structures are defined. Typically, these tentative designs are characterized by a set of parameters and some sort of experimental design [1] is applied to determine the *best* parameters' values;
(d) experimentation: the tentative neighborhood structures are assessed on a sample (*training set*) extracted from the reference instance population. The results obtained in this phase may suggest some modifications of the tentative neighborhood structures.

The aim of our research is to develop, without any human intervention, "good" metaheuristics from a given Mixed Integer Programming (MIP) model. The ultimate goal is to generate automatically metaheuristics that may provide *human competitive* results [2]. Recently [3] introduced a three-step procedure, which automatically designs a single neighborhood structure from a MIP model: (1) a set of semantic features are automatically extracted from both the MIP model and a given feasible solution; (2) neighborhood design mechanisms are derived from the extracted features; (3) a "proper mix" of such mechanisms are searched during an automatic configuration phase. In this paper we take a different perspective and generalize the previous work in the context of an entire metaheuristic algorithm. More specifically, we focus on *Variable Neighborhood Descent* (VND) [4] and define a procedure that - based on a MIP model and a current feasible solution - determines automatically the size and the "shape" of the whole VND hierarchy of neighborhoods, as well as the time limits for their exploration through a general-purpose black-box MIP solver.

The remainder of the paper is organized as follows. Section 2 is devoted to a review of the literature relevant to our work. In

* Corresponding author.
E-mail addresses: tommaso.adamo@unisalento.it (T. Adamo), gianpaolo.ghiani@unisalento.it (G. Ghiani), emanuela.guerriero@unisalento.it (E. Guerriero), emanuele.manni@unisalento.it (E. Manni).

https://doi.org/10.1016/j.orp.2017.09.001
2214-7160/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license. (http://creativecommons.org/licenses/by-nc-nd/4.0/)

Section 3 we present the basic idea underlying the proposed approach and describe its main procedures. In Section 4 we discuss the computational results obtained on four well-known combinatorial optimization problems. In particular, we discuss the improvements provided by our automatically-designed VND with respect to both a previous model-derived Variable Neighborhood Descent procedure and the approach with a single automatically-designed neighborhood structure. Finally, conclusions follow in Section 5.

2. Literature review

Our work is related to several areas. First of all, it is related to the field of model-derived neighborhoods. Among these contributions, very relevant are those based on the *Local Branching* concept [5] in which spherical neighborhoods defined by appropriate non valid inequalities are explored by using an off-the-shelf MIP solver. In particular, for purely binary MIPs a neighborhood of the current solution includes all the solutions in which the number of variables changing value (i.e., the *Hamming distance*) does not exceed a given threshold. Danna et al. [6] introduced the *Relaxation-Induced Neighborhood*, which is defined by fixing the variables with the same values in both the incumbent and the optimal solution of the continuous relaxation. Then, after setting a cutoff equal to the objective value of the current incumbent, the neighborhood is explored by solving the sub-MIP on the remaining variables. Parisini and Milano [7] presented a search strategy called *Sliced Neighborhood Search* that considers randomly selected slices of spherical neighborhoods. Particularly relevant to our work are the contributions by Ghiani et al. [8] and Adamo et al. [3], that showed how to take advantage of a MIP compact formulation to automatically design efficient neighborhood structures, without any human analysis. In particular, Ghiani et al. [8] used unsupervised learning to automatically select "good" portions of the search space "around" a given feasible solution. Adamo et al. [3] proposed a procedure extracting some semantic features from a given MIP model. Based on the selected features, some neighborhood design mechanisms are automatically derived and, finally, a "proper mix" of such mechanisms are determined by running an automatic configuration algorithm on a training set representative of the reference instance population. This approach was recently extended by Adamo et al. [9], that allowed the Automatic Neighborhood Design algorithm to deal with an *ensemble* of Constraint Programming (CP) and MIP models.

Another area to which our paper is strongly related is that combining model-derived neighborhoods and heuristic search. Such approaches are typically referred to as *matheuristics* [10], which are defined as heuristic algorithms obtained by integrating metaheuristics and mathematical programming techniques. In particular, Hansen et al. [11] combined Local Branching with *Variable Neighborhood Search*, whereas Lazić et al. [12] proposed to solve 0–1 MIPs by a hybrid heuristic based on the principle of *Variable Neighborhood Decomposition Search*. Then, [13] devised a general matheuristic that decomposes the problem being solved in a master and a subproblem. The main characteristic of the approach is that it exploits features of the incumbent solution to generate one or more columns in the master problem. More recently, [14] proposed a general approach for combinatorial optimization termed *Construct, Merge, Solve & Adapt*, in which sub-instances of the original problem are first generated by repeatedly constructing probabilistic solutions and then solved by using a general-purpose MIP solver. In the context of CP, Van Hentenryck and Michel [15] showed how constraint-based local search algorithms can be synthesized from high-level models, at least for some application classes. Such synthesis is driven by the model structure, as well as the role and the semantics of each single constraint. In particular, the high-level model is first classified and then a solution algorithm is selected from a predefined portfolio. These ideas have been exploited by Elsayed and Michel [16] to develop a model-driven automatic search procedure generator written in Comet [17]. The generator examines a CP model instance, analyzes the constraints as well as the variable declarations and synthesizes a procedure that is likely to yield good performances on the considered instance. More recently, Mouthuy et al. [18] showed how these concepts can be applied to a *Very Large Scale Neighborhood Search* framework, whereas Kiziltan et al. [19] combined constraint propagation with the Local Branching general-purpose neighborhood.

An alternative approach involves the use of hyper-heuristics [20] in which, given a particular problem instance, an appropriate low-level heuristic is selected from a given set and applied at each step. For instance, in the context of genetic programming [21] developed a system that uses a simple composition operator to automatically discover local search heuristics for the boolean satisfiability testing problem. A recent research trend [22,23] is to employ human-designed heuristics as building-blocks to automatically generate new heuristics suited to a given problem or class of problems.

Finally, our work is also related to *Automatic Algorithm Configuration* (AAC) [24–26], in which an automatic procedure finds the parameter configurations for which the empirical performance on a given set of problem instances is optimized. Nowadays, many AAC software packages have been developed, such as F-Race [27,28], Calibra [29], ParamILS [30] and irace [31]. AAC has also been used in combination with grammar representations, as in [32], where the authors proposed a novel representation of the grammar by a sequence of categorical, integer, and real-valued parameters. Then, they used an AAC tool to search for the best algorithm for the problem at hand.

3. Automatic instantiation of a variable neighborhood descent

As stated before, the aim of our research is to develop automatically "good" metaheuristics from a MIP model. The basic idea is to try to reproduce the behavior of a human researcher that must develop a neighborhood search heuristic for a given combinatorial optimization problem $\mathcal{P}$. The starting point is a knowledge base of the problem, that is thoroughly examined in order to identify the main components that are combined to obtain a feasible solution. For instance, in a vehicle routing problem such components would be the customers and the vehicles that must be associated each other to define the routes. Of course, the solution is feasible if customers to vehicles assignment satisfies some requirements (constraints) that are identified from the problem description. After defining the structure of a feasible solution, the researcher must identify neighborhood relationships on the search space that allow to move from a current solution to a new (possibly improving) one. For instance, in a vehicle routing problem a new solution could be obtained by moving a given number of customers from a route to another.

Analogously to a human-like approach, we must define a way to represent the knowledge associated to the problem that allows to easily and automatically identify and extract the main components of the problem as well as the relationships among them (*semantic features*, in the following). In this paper, we assume that $\mathcal{P}$ is given a *factored* representation (in which each state is coded by a fixed set of variables, see [33]) as a mixed-integer program. Let $J = \mathcal{C} \cup \mathcal{G} \cup \mathcal{B} = \{1, \ldots, n\}$ be the variable index set ($\mathcal{C}$, $\mathcal{G}$, and $\mathcal{B}$ denote the index sets for continuous, general integer and binary variables, respectively). A generic MIP model for $\mathcal{P}$ can be stated

as:

$$(\mathcal{P}) \quad \min z = \sum_{j=1}^{n} c_j x_j \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j = b_i, i = 1, \ldots, m \tag{2}$$

$$x_j \geq 0, j \in \mathcal{C} \tag{3}$$

$$x_j \geq 0 \text{ and integer}, j \in \mathcal{G} \tag{4}$$

$$x_j \in \{0, 1\}, j \in \mathcal{B} \neq \emptyset. \tag{5}$$

However, this kind of representation does not give any information about the problem components (customers and vehicles in the previous VRP example) and their relationships. Thus, we need a representation that is closer to what humans do, namely a *structured* representation (in which objects and their various and varying relationships can be described explicitly, see [33]). As [33] note: "In fact, almost everything that humans express in natural language concerns objects and their relationships". For this reason, after writing the MIP model for $\mathcal{P}$, we require that it is coded through an algebraic modeling language (e.g., AMPL [34], GAMS [35] or OPL [36]). Such languages rely on sets of objects (referred to as *entities* from now on, to be consistent with the notation of [3], and as also very common in the context of computer science [37]) such as customers, vehicles, or facilities. Entities are classified as *fundamental* or *derived*, where a derived set is defined as subsets or Cartesian products of other sets. As will be described later, our instantiation mechanisms are based on the information related to fundamental entities only. Then, each variable, constraint and parameter of the MIP model is *indexed* by one or more entities in the corresponding algebraic modeling. The information about entities and their relationships are automatically detected and extracted by using a parser that is based on the syntax and the grammar of the algebraic modeling language used to encode the model.

3.1. Automatic extraction of semantic features

The indexing mechanism illustrated before is key to identify semantic relationships between entities that are necessary for the proper design of our neighborhood structures. In this section, we detail the procedure used to extract the semantic features from both the MIP model and a given feasible MIP solution.

Formally, we denote as $E = E_1 \cup \cdots \cup E_K$ the set of fundamental entities characterizing our model, where E_k $(k = 1, \ldots, K)$ is a subset of homogeneous entities, such as a set of vehicles, a set of customers, a set of commodities, etc. The semantic features can be of two types: *model-based* features, derived directly from an instance of the MIP model, and *solution-based* features, identified from a given current feasible solution.

With respect to the model-based features, we denote by p_k the number of parameters indexed by entities in the set E_k, with $k = 1, \ldots, K$ (e.g., in the generic MIP formulation (1)–(5), the parameters of the model are a_{ij}, b_i, c_j, with $i \in 1, \ldots, m$ and $j \in 1, \ldots, n$). Then, we define a dataset $\mathcal{D}_k$ where each column (or group of columns) refers to one of the p_k parameters, while each observation (i.e., row) is associated with an entity in E_k. To take into account that some features can be more relevant than others, we associate a weight w_l^k to the l-th parameter $(l = 1, \ldots, p_k; k = 1, \ldots, K)$. After normalizing the weighted dataset, the model-based similarity between two entities $e, e' \in E_k$ is computed as the inner product of the corresponding rows. We observe that, as in [3], the most appropriate values of all weights w_l^k $(l = 1, \ldots, p_k$ and $k = 1, \ldots, K)$ are sought through an automatic configuration phase performed on a training set (representative of the reference instance population and separated from the set of instances used during the test phase).

In addition, the strength of the relationships between two entities $e, e' \in E_k$ is measured with respect to the current solution. First of all, given a current feasible solution $\mathbf{x}'$, we define as *adjacent* two entities $e, e' \in E$ such that: (i) they are both indexing a variable having a nonzero value in $\mathbf{x}'$, or (ii) there is a constraint indexed by e that is active in $\mathbf{x}'$ (i.e., the corresponding auxiliary variable used to standardize (2) is zero), in which a variable, indexed by e', appears with a nonzero coefficient and has a nonzero value in $\mathbf{x}'$ (or vice versa). Such an adjacency is used to define the *Entity Adjacency Graph* (EAG), that is a graph where the vertex set is made up of a node for each entity in E. The objective function z is treated as a special entity that is present in constraint (1). In this way, z is considered adjacent to the entities having a direct impact on it, i.e., the entities that index the variables appearing in (1). Moreover, an arc is included in the EAG for each pair of adjacent entities in the vertex set. The EAG plays a fundamental role in the definition of good neighborhood structures.

In the following, we illustrate the previous concepts on a classic combinatorial optimization problem that is first modeled as a MIP and then written in the OPL algebraic modeling language. For this problem, we first show how to build the dataset used to identify the model-based features and then how to construct the EAG.

Example. As an example of combinatorial optimization problem, we consider the well-known Traveling Salesman Problem (TSP), which is one of the test problems used in Section 4. The TSP amounts to determine a minimum cost Hamiltonian circuit on a complete directed graph $G = (N, A)$, in which $N = \{1, \ldots, n\}$ is the vertex set (representative of the customers to visit) and $A = \{(i, j) : i, j \in N; i \neq j\}$ is the set of arcs connecting the vertices. Each arc (i, j) has an associated cost c_{ij} for traversing it. A possible MIP formulation [38] is:

$$\min z = \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{6}$$

$$\text{s.t.} \sum_{j \in N} x_{ij} = 1 \qquad i \in N, \tag{7}$$

$$\sum_{i \in N} x_{ij} = 1 \qquad j \in N, \tag{8}$$

$$u_i - u_j + 1 \leq (n-1)(1 - x_{ij}) \qquad i, j \in N, i, j \neq 1, \tag{9}$$

$$u_1 = 1, \tag{10}$$

$$2 \leq u_i \leq n \qquad i \in N, i \neq 1, \tag{11}$$

$$x_{ij} \in \{0, 1\} \qquad (i, j) \in A, \tag{12}$$

where x_{ij}, $(i, j) \in A$, is equal to 1 if arc (i, j) is in the tour, otherwise its value is 0. Variables u_i, $i \in N$, are used for sub-tours elimination.

A possible OPL representation of formulation (6)–(12) is showed in Fig. 1. Here, the entities are the customers (set `CUSTOMERS`) and the arcs (set `ARCS`). In particular, the customers are fundamental entities, whereas the arcs are derived, being a subset of the Cartesian product of the set `CUSTOMERS`. Formally, $K = 1$ (the customers), $E = E_1 = N$ and $p_1 = 1$ (the cost matrix). Variables `x` and `u`, as well as the parameter `cost`, are indexed by the customers.

```
1  int n = ...;
2  {int} CUSTOMERS = {i | i in 1..n};
3  int cost[CUSTOMERS][CUSTOMERS] = ...;
4  tuple Arc {key int i; key int j;};
5  {Arc} ARCS = {<i,j> | i, j in CUSTOMERS: i != j};
6  dvar boolean x[ARCS];
7  dvar float+ u[CUSTOMERS];
8  minimize sum(<i, j> in ARCS) cost[i][j] * x[<i,j>];
9  subject to {
10 forall(i in CUSTOMERS) sum(j in CUSTOMERS: <i,j> in ARCS) x[<i,j>] == 1;
11 forall(j in CUSTOMERS) sum(i in CUSTOMERS: <i,j> in ARCS) x[<i,j>] == 1;
12 forall(<i,j> in ARCS: i != 1 && j != 1)
13    u[i] - u[j] + 1 <= (n-1) * (1 - x[<i,j>]);
14 forall(i in CUSTOMERS: i != 1) u[i] <= n;
15 forall(i in CUSTOMERS : i != 1) u[i] >= 2;
16 u[1] == 1;
17 };
```

Fig. 1. OPL representation for the Traveling Salesman Problem.

Table 1
Dataset used to compute the model-based similarity for the Traveling Salesman Problem.

Entity	Parameters					
1	c_{11}	...	c_{1n}	c_{11}	...	c_{n1}
...	...	...	...	...	...	...
i	c_{i1}	...	c_{in}	c_{1i}	...	c_{ni}
...	...	...	...	...	...	...
n	c_{n1}	...	c_{nn}	c_{1n}	...	c_{nn}

On the other hand, all the constraints are indexed by the entities in set CUSTOMERS, with the only exception of constraint reported at line 16 of the OPL model.

In Table 1 we report the dataset D_1 that is characterized by n rows ($|E_1| = n$). The ith row (associated to the ith customer) contains the values of all the parameters indexed by i (i.e., all the elements of cost that are related to customer i). In the weighted dataset the ith row will be weighted by w_1^1.

With respect to solution-based features, the EAG is constructed as follows. The set of vertices contains a vertex representative of z, as well as a vertex for each customer in E_1. In particular, given a feasible solution z is adjacent to each entity in E_1 (in a feasible TSP solution each customer must be visited) and each customer is also adjacent to the customers visited just before and after it. The latter information is obtained from: (i) variables x_{ij} that take value 1 when i and j are consecutive; (ii) constraints (7)–(9). A graphical representation of the EAG is reported in Fig. 2, where the dotted line is used to indicate that the vertex representing z is connected with all the vertices associated with customers.

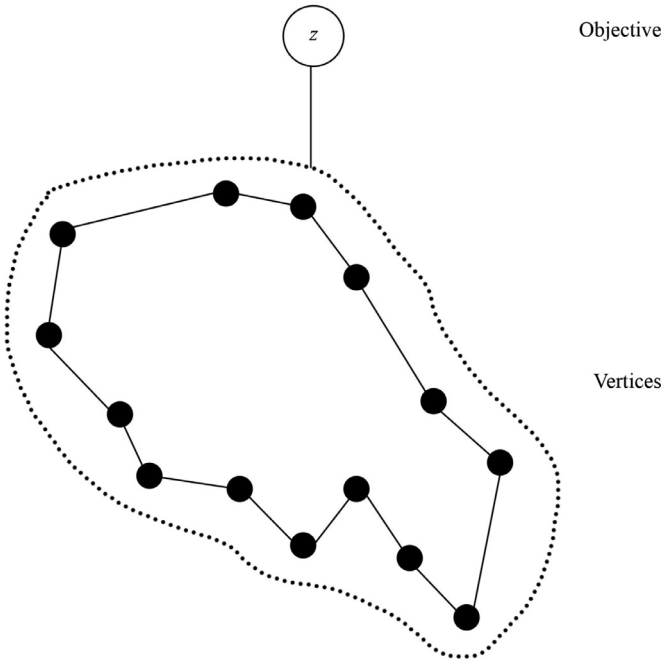

Fig. 2. Example of an Entity Adjacency Graph for a formulation of the Traveling Salesman Problem. The dotted line indicates that the vertex representing z is connected with all the vertices associated with customers.

3.2. Neighborhood structure design

In order to define a neighborhood structure, the first step is to identify a fragment of the current solution $\mathbf{x}'$ indexed by entities in a set $F \subseteq E$ that are classified as adjacent with respect to $\mathbf{x}'$, or similar with respect to the model. Then, a neighborhood $\mathcal{N}(\mathbf{x}')$ is defined as the set of feasible solutions obtained by first destroying and then repairing the selected fragment of $\mathbf{x}'$ indexed by the entities in F. Formally, given an entity $e \in E$, let $J_e \subseteq J$ denote the set of indices associated with variables indexed by e. We model the

neighborhood structure $\mathcal{N}(\mathbf{x}')$ as the feasible region of the sub-problem:

$$(\mathcal{P}(\mathbf{x}', F)) \qquad \min \sum_{j \in \bigcup_{e \in F} J_e} c_j x_j + \sum_{j \in J \setminus \bigcup_{e \in F} J_e} c_j x'_j \tag{13}$$

s.t. (2)-(5), and

$$x_j = x'_j, \quad j \in J \setminus \bigcup_{e \in F} J_e. \tag{14}$$

Problems (13)–(14) are obtained from the original problems (1)–(5) by adding the fixing constraint (14). The aim of this constraint is to consider as fixed to their values in $\mathbf{x}'$ all the variables indexed by entities in F. On the other hand, the remaining variables can vary in their original domains. Then, the repair is done by determining the optimal solution of sub-problem (13)–(14) by means of an off-the-shelf MIP solver with a given parameter setting.

The key aspect is how to define a 'good' neighborhood structure, i.e., a 'good mix' of similar and adjacent entities to include in F. The procedure we employ is the same as in [3] and is composed of four steps, that are reported here for the sake of completeness:

1. Select a subset F' of entities adjacent in $\mathbf{x}'$ and making up a connected sub-graph of the EAG adjacent to z. In particular the entities are selected on the EAG by performing a random walk starting from z. This process is performed until the number of variables indexed by entities in F' and having a nonzero value in $\mathbf{x}'$ becomes greater than a threshold, obtained as a fraction α' of the overall number of variables. At each step of the definition of the connected sub-graph, an entity in E_k $(k = 1, \ldots, K)$ is selected with probability π_k $(k = 1, \ldots, K)$;
2. Select a subset F'' among the remaining entities that are most *similar* to those in F', according to the definition of Section 3.1. Analogously to step 1, we add entities to F'' until the cumulative number of variables indexed by them and having a nonzero value in $\mathbf{x}'$ is lower than or equal to a fraction α'' of the overall number of variables;
3. Select a subset F''' of entities not yet selected and adjacent in $\mathbf{x}'$ to entities in F''. Again, the cardinality of F''' is such that the number of variables indexed by its entities and having a nonzero value in $\mathbf{x}'$ is lower than a fraction α''' of the overall number of variables.
4. Set $F = F' \cup F'' \cup F'''$.

It is worth noting that the size of neighborhood $\mathcal{N}(\mathbf{x}')$ is determined by the number of variables indexed by entities in F having a nonzero value in $\mathbf{x}'$. In particular, the neighborhood $\mathcal{N}(\mathbf{x}')$ corresponding to (13)–(14) must be *sufficiently small* to be explored in a reasonable computing time, but still *large enough* to likely contain better solutions than $\mathbf{x}'$. For these reasons, it is crucial to give appropriate values to the fractions α', α'', and α'''. Thus, the choice of such values, as well as those of the other parameters of the procedure, is performed through an AAC phase by using a state-of-the-art tool. In particular, the parameters to be tuned are:

- the upper bounds α', α'', and α''' selected at steps 1, 2, and 3, respectively;
- the weights w_l^k $(l = 1, \ldots, p_k; k = 1, \ldots, K)$ needed to define the model-based similarities among entities used in Step 2;
- the probabilities π_k $(k = 1, \ldots, K)$ of visiting an entity belonging to E_k $(k = 1, \ldots, K)$ in Step 1.

Moreover, since our procedure aims to find the "best" neighborhood structures that optimize the empirical performance of a VND on the peculiar distribution of the instances to be solved, the user must provide a training set representing the reference instance population. Such training set is used by the AAC procedure to determine the most appropriate values for the parameters (also called *design variables*).

3.3. Variable neighborhood descent instantiation

As a step towards the automatic instantiation of complex metaheuristic algorithms, the main contribution of this paper is to extend to a VND algorithm the approach previously developed by [3] for a single neighborhood structure.

The VND pseudocode is as reported in Algorithm 1. Given a current solution $\mathbf{x}'$, a finite number of $r_{\max}$ pre-selected neighborhood structures $\mathcal{N}_r$ $(r = 1, \ldots, r_{\max})$, and a time limit t_r for each of them, the VND procedure systematically switches between the different neighborhood structures. A neighborhood structure $\mathcal{N}_r$ $(r = 1, \ldots, r_{\max})$ is defined by the set of parameters described at the end of Section 3.2 that, together with $\mathbf{x}'$, allow to obtain a set F. In order to highlight the dependence on r, in the following we will refer to F as F_r, and to α', α'' and α''' as α'_r, α''_r and α'''_r.

Algorithm 1 Pseudocode of a Variable Neighborhood Descent procedure.

```
1: procedure VND(x', N_1, ..., N_rmax, t_1, ..., t_rmax)
2:     r ← 1
3:     while r ≤ r_max do
4:         x'' ← NeighborhoodExploration(x', N_r, t_r)
5:         x', r ← NeighborhoodChange(x', x'', r)
6:     end while
7:     return x'
8: end procedure
```

Starting from the first structure $\mathcal{N}_1$, VND performs a local search phase (procedure NeighborhoodExploration, for which a pseudocode is illustrated in Algorithm 2) until a local optimum is reached or the time t_1 at disposal is consumed. From this new current solution (which, of course, could be the same as the previous one), it continues the local search phase with $\mathcal{N}_1$ if the objective function has been improved, otherwise it switches to $\mathcal{N}_2$. If an improved solution can be found with this structure, VND returns to using $\mathcal{N}_1$ again; otherwise, it continues with $\mathcal{N}_3$, and so forth. If $\mathcal{N}_{r_{\max}}$ has been explored and no further improvements are possible, then the VND terminates. Function NeighborhoodChange (a pseudocode is reported in Algorithm 3) is in charge of comparing the objective value of the incumbent with that of the new solution

Algorithm 2 Pseudocode of the NeighborhoodExploration procedure.

```
1: procedure NeighborhoodExploration(x', N_r, t_r)
2:     F_r ← < Select entities according to N_r >
3:     x'' ← < Solve (P(x', F_r)) with time limit t_r >
4:     return x''
5: end procedure
```

Algorithm 3 Pseudocode of the NeighborhoodChange procedure.

```
1: procedure NeighborhoodChange(x', x'', r)
2:     if Σ_{j=1}^{n} c_j x''_j < Σ_{j=1}^{n} c_j x'_j then
3:         return 1
4:     else
5:         return r + 1
6:     end if
7: end procedure
```

obtained by exploring the current neighborhood and switching between the different neighborhood structures.

In our VND implementation, the hierarchy of $r_{\max}$ neighborhood structures is obtained by varying the size of F_r, by considering different values for α'_r, α''_r and α'''_r. In particular, the r-th neighborhood ($r = 1, \ldots, r_{\max}$) is explored by solving - through a black-box MIP solver - the following problem:

$$(P(\mathbf{x}', F_r)) \qquad \min \sum_{j \in \bigcup_{e \in F_r} J_e} c_j x_j + \sum_{j \in J \setminus \bigcup_{e \in F_r} J_e} c_j x'_j \tag{15}$$

s.t. (2)–(5), and

$$x_j = x'_j, \quad j \in J \setminus \bigcup_{e \in F_r} J_e. \tag{16}$$

As a result, the size and shape of the rth neighborhood structure ($r = 1, \ldots, r_{\max}$) depends on set F_r, whose cardinality - as mentioned before - is influenced by the threshold values α'_r, α''_r and α'''_r. Thus, the design problem amounts to determine - through an AAC procedure - the size and the time limit for each of the VND neighborhoods, as well as the other parameters reported at the end of Section 3.2.

4. Computational results

We have tested our approach on four classical combinatorial optimization problems, namely the *Traveling Salesman Problem* (TSP), the *Generalized Traveling Salesman Problem* (GTSP), the *Capacitated Vehicle Routing Problem with Time Windows* (VRPTW), and the *Multi-Plant, Multi-Item, Multi-Period Capacitated Lot-Sizing Problem* (MPCLSP). For each of these test problems, the aim of our computational experiments is to compare the performance of: (i) our automatically-designed VND (AD-VND) made up of a hierarchy of $r_{\max}$ automatically-designed neighborhoods of different sizes and shapes; (ii) a model-based VND using a hierarchy of $r_{\max}$ spherical neighborhoods (SN-VND) of different sizes. Moreover, to measure the benefit of the VND with respect to the usage of a single, tuned neighborhood size, we also compare to the single automatically-designed neighborhood (ADN) of [3].

In the case of AD-VND, we consider $r_{\max} = 3$. Then, given an initial feasible solution $\mathbf{x}'$, the r-th neighborhood is explored by solving problems $(P(\mathbf{x}', F_r))$ $(r = 1, 2, 3)$ as detailed in Section 3.3. In this case, the design variables, whose values are obtained by an AAC procedure, are: the time limits t_1, t_2 and t_3; the thresholds α'_r, α''_r and α'''_r influencing the size of sets F_r $(r = 1, 2, 3)$; the probabilities π_k $(k = 1, \ldots, K)$; the weights w_l^k given to the parameters of the MIP models $(l = 1, \ldots, p_k;\ k = 1, \ldots, K)$. The most appropriate values of the design variables identified by the AAC algorithm are marked with an asterisk in the following.

With respect to the model-based VND using spherical neighborhoods, as before we consider $r_{\max} = 3$. Given an initial feasible solution $\mathbf{x}'$, the rth spherical neighborhood [5] is explored by solving - through a black-box MIP solver - model (1)–(5) together with constraint

$$\sum_{j \in \mathcal{B}:\, x'_j = 1} (1 - x_j) + \sum_{j \in \mathcal{B}:\, x'_j = 0} x_j \leq \Delta_r, \tag{17}$$

and imposing a time limit equal to t_r. The most appropriate values for the sizes Δ_1, Δ_2, Δ_3 and the time limits t_1, t_2, t_3 are chosen by an AAC procedure and are marked with an asterisk in the following.

Finally, in the case of ADN we have considered the same parameter settings as in [3].

For the three approaches, the neighborhoods are explored by using the black-box solver IBM ILOG CPLEX 12.6 (with thread-concurrency enabled), while OPL and ParamILS are used as mathematical programming language and automatic parameter tuner, respectively. For each test problem, we have devoted the 40% of the reference instance population for the training phase, while the remaining 60% has been considered as test set. We observe that, to allow a fair comparison, the automatic parameter tuning procedure and the effort are the same used by [3]. The parser used to analyze the structure of the OPL encoding of the MIP models is implemented by using the tool ANTLR [39]. All the experiments have been performed on a Linux machine equipped with an Intel Xeon CPU X5550 clocked at 2.67 GHz (cache size 8192 KB), and 27 GB of RAM. Moreover, we have imposed an overall time limit of 20 min for the execution of the three tested approaches, whereas an overall time limit of 24 h has been devoted to the tuning phase, with a limit of 20 min for each run.

For each test problem, we report:

- the objective function value z_0 of the initial feasible solution (when describing the different problems, we also illustrate how such solutions are obtained);
- the solution value z of the best solution found in each approach (ADN, SN-VND and AD-VND) as well as the percentage improvement DEV_{z_0} with respect to z_0.

Moreover, when the optimal solution is known, as in the TSP and the GTSP, we report the percentage deviation (DEV_{opt}) of z_0 from the optimum. Otherwise, we report the percentage optimality gap (GAP) of z_0 as determined by the MIP solver, that is, $100 \cdot |\texttt{bestnode} - z_0| / (10^{-10} + |z_0|)$, where `bestnode` is the objective function value of the best node found by the MIP solver when inputted with z_0. We observe that GAP is an upper-bound (i.e., an over-estimation) of DEV_{opt}.

We do not present any result about the computing time, because the time limit of 20 min has always been reached. We also report the mean value of the percentage improvement (MEAN) as well as the width of the 95% confidence interval around the mean (CI). Finally, in each table bold numbers represent the best results for a given instance or class of instances.

4.1. Traveling Salesman Problem

For the TSP we have used the formulation of Section 3.1. Such a formulation contains just one type ($K = 1$) of fundamental entities (the vertices) that are characterized by a single parameter (the cost matrix). Hence, $p_1 = 1$ and $\pi_1 = 1$. Therefore, the design variables for AD-VND are α'_r, α''_r, α'''_r and t_r $(r = 1, 2, 3)$. In particular, the sets of possible values that we consider are {0.05, 0.10, 0.15} for α'_1, {0.15, 0.20, 0.25} for α'_2, {0.25, 0.30, 0.35} for α'_3, {0.000, 0.025, 0.050, 0.075, 0.100, 0.0125, 0.150, 0.175, 0.200} for α''_r and α'''_r $(r = 1, 2, 3)$, and {0, 1} for w_1^1. On the other hand, the only design variables for SN-VND are Δ_r and t_r $(r = 1, 2, 3)$ with values chosen in $\{n/256, n/128, n/64\}$ for Δ_1, $\{n/64, n/32, n/16\}$ for Δ_2 and $\{n/16, n/8, n/4\}$ for Δ_3. With respect to the time limits, their values are chosen for both approaches in {60, 120, 180, 240, 300, 360} s.

The values of the design variables for the AD-VND, identified after the training phase, are: $\pi_1^* = 1$, $w_1^{1*} = 0$, $\alpha_1'^* = 0.10$, $t_1^* = 120$ s, $\alpha_2'^* = 0.20$, $t_2^* = 240$ s, $\alpha_3'^* = 0.30$, $t_3^* = 360$ s, $\alpha_r''^* = 0.00$ $(r = 1, 2, 3)$ and $\alpha_r'''^* = 0.00$ $(r = 1, 2, 3)$. The meaning of $w_1^{1*} = 0$, $\alpha_r''^* = 0.00$ $(r = 1, 2, 3)$ and $\alpha_r'''^* = 0.00$ $(r = 1, 2, 3)$ is that the sets F_r $(r = 1, 2, 3)$ coincide with sets F'_r, i.e., the neighborhood structures are directly influenced only by the solution-based features.

With respect to the parameters of the SN-VND, their values after the training phase are: $\Delta_1^* = n/64$, $t_1^* = 120$ s, $\Delta_2^* = n/32$, $t_2^* = 240$ s, $\Delta_3^* = n/16$ and $t_3^* = 300$ s.

The two approaches have been tested on instances based on Euclidean distance taken from the TSPLIB [40] for which initial

Table 2
Results for the TSP.

Instance	z_0	DEV_{opt} (%)	ADN		SN-VND		AD-VND	
			z	DEV_{z_0} (%)	z	DEV_{z_0} (%)	z	DEV_{z_0} (%)
lin318	54,019	22.20	50,615	6.30	49,241	8.85	49,116	**9.08**
linhp318	54,019	23.46	50,615	6.30	49,152	9.01	49,116	**9.08**
rd400	19,183	20.34	17,452	9.02	16,816	**12.34**	17,314	9.74
fl417	15,013	21.00	14,012	6.67	12,868	14.29	12,860	**14.34**
pr439	131,281	18.33	125,756	4.21	112,537	**14.28**	116,275	11.43
pcb442	61,979	18.07	59,389	4.18	53,933	**12.98**	55,084	11.12
d493	41,665	15.99	38,945	6.53	38,104	**8.55**	38,773	6.94
u574	50,459	26.86	48,863	3.16	44,576	11.66	44,053	**12.70**
rat575	8605	21.29	8126	5.57	7932	**7.82**	8052	6.43
p654	43,457	20.28	39,188	9.82	38,343	**11.77**	38,723	10.89
d657	61,627	20.63	56,493	8.33	56,051	9.05	55,973	**9.17**
u724	52,943	20.84	52,093	1.61	49,176	**7.12**	50,031	5.50
rat783	11,054	20.34	10,479	5.20	10,906	1.34	10,207	**7.66**
pr1002	331,103	21.76	322,161	2.70	–	–	322,039	**2.74**
u1060	308,980	27.47	295,620	4.32	–	–	292,541	**5.32**
vm1084	301,477	20.63	292,357	3.03	–	–	279,636	**7.24**
pcb1173	71,978	20.96	70,434	2.15	–	–	69,082	**4.02**
d1291	60,214	15.63	59,731	0.80	–	–	58,833	**2.29**
rl1304	335,779	24.67	331,268	1.34	–	–	330,404	**1.60**
nrw1379	68,964	17.87	68,086	1.27	–	–	66,760	**3.20**
fl1400	27,447	26.67	26,513	3.40	–	–	26,396	**3.83**
u1432	188,807	18.98	181,503	3.87	–	–	181,421	**3.91**
d1655	74,033	16.08	73,210	1.11	–	–	73,211	**1.11**
u1817	72,030	20.59	71,293	1.02	–	–	70,950	**1.50**
u2152	79,260	18.93	78,778	0.61	–	–	78,646	**0.77**
u2319	278,765	15.97	275,999	0.99	–	–	274,522	**1.52**
pcb3038	176,310	21.90	175,974	0.19	–	–	174,445	**1.06**
MEAN				3.84		4.78		**6.08**
CI				1.03		1.92		1.52

solutions are obtained by choosing a first vertex at random, selecting its nearest neighbor, and iterating the procedure until all the vertices are added to the tour. Table 2 reports the comparison of the obtained results. First of all, it is worth noting that, as the size of the instances grows (in particular, for a number of vertices greater than 783) the SN-VND is not even able to load the model, while this is not the case of AD-VND and ADN. The explanation to this behavior is that the variable fixing introduced by constraints (14) results in a model of a reduced size with respect to the original model. Analyzing in detail the results, the average SN-VND improvement is almost 5%, while the AD-VND provides an average improvement of about 6%. In this case, the main advantage of AD-VND is its scalability, allowing to provide improvements (although quite small in some case) even for very large instances. With respect to the comparison with ADN, using a hierarchy of neighborhoods always allows to achieve better results, with values of DEV_{z_0} that in some cases are even four times higher.

4.2. Generalized Traveling Salesman Problem

Let us consider a complete directed graph $G = (N, A)$, where $N = \{1, \ldots, n\}$ is the vertex set and $A = \{(i, j) : i, j \in N; i \neq j\}$ is the set of arcs. N is partitioned into H clusters N_h $(h = 1, \ldots, H)$. The travel cost from i to j is denoted with c_{ij}, whereas the binary coefficient a_{ih}, $i \in N$, $h \in \{1, \ldots, H\}$ has value 1 if node i belongs to cluster h, 0 otherwise. The GTSP aims to find a minimum-cost cycle that visits exactly one node from each cluster. The formulation is the same used in Adamo et al. [3] and is reported for completeness:

$$\min z = \sum_{(i,j)\in A} c_{ij}x_{ij}$$

$$\text{s.t.} \sum_{j\in N\setminus\{i\}} x_{ij} = z_i, \qquad i \in N,$$

$$\sum_{j\in N\setminus\{i\}} x_{ji} = z_i, \qquad i \in N,$$

$$\sum_{i\in N} a_{ih}z_i = 1, \qquad h = 1, \ldots, H,$$

$$u_i - u_j + nx_{ij} \leq n - 1, \qquad i, j \in N \quad i, j \geq 1 \quad i \neq j,$$

$$y_i - y_j + 1 - \sum_{h=1}^{H} a_{ih}a_{jh} \leq (H + 1)(1 - x_{ij}), \qquad i, j \in N, j \neq 0,$$

$$1 \leq u_i \leq n, \qquad i \in N \setminus \{0\},$$

$$1 \leq y_i \leq H, \qquad i \in N \setminus \{0\},$$

$$u_0 = y_0 = 0,$$

$$z_0 = 1,$$

$$z_i \in \{0, 1\}, \qquad i \in N \setminus \{0\},$$

$$x_{ij} \in \{0, 1\} \qquad (i, j) \in A.$$

The binary variables x_{ij}, $(i, j) \in A$, take value 1 only if arc (i, j) is in the tour. Variables y_i, u_i, $i \in N$, are used to exclude clusters sub-tours and node sub-tours, respectively. Moreover z_i, $i \in N$, models the decision of including a node in the circuit or not.

For this problem, the clusters and the arcs are considered derived entities. In fact, the clusters are obtained as subsets of the vertices, whereas the arcs, as in the case of the TSP, are a subset of the Cartesian product of the vertices. Thus, there is only one type $(K = 1)$ of fundamental entities (the vertices) characterized by a dataset $\mathcal{D}_1$ with two attributes: arc cost c_{ij} $(l = 1)$ and the binary coefficient a_{ih} $(l = 2)$.

Hence, $p_1 = 2$ and it $\pi_1 = 1$. Therefore, the design variables for AD-VND are: α'_r, α''_r, α'''_r, t_r $(r = 1, 2, 3)$, w_1^1 and w_2^1. The sets of possible values that we consider are {0.05, 0.10, 0.15} for α'_1, {0.15, 0.20, 0.25} for α'_2, {0.25, 0.30, 0.35} for α'_3, {0.000, 0.025, 0.050, 0.075, 0.100, 0.0125, 0.150, 0.175, 0.200} for α''_r and α'''_r $(r = 1, 2, 3)$, and {0, 1} for w_1^1 and w_2^1. On the other hand, the design vari-

Table 3
Results for the GTSP.

Instance	z_0	DEV_{opt} (%)	ADN		SN-VND		AD-VND	
			z	DEV_{z_0} (%)	z	DEV_{z_0} (%)	z	DEV_{z_0} (%)
64lin318	23,552	17.22	20,533	12.82	21,516	8.64	20,022	**14.99**
80rd400	7826	20.93	6743	13.84	7509	4.05	6722	**14.11**
84fl417	9246	5.91	8799	4.83	9197	0.53	8787	**4.96**
88pr439	67,084	16.02	59,002	12.05	63,481	5.37	57,160	**14.79**
89pcb442	27,358	23.85	22,781	16.73	24,830	9.24	22,560	**17.54**
99d493	21,537	16.35	19,675	8.65	21,537	0.00	19,544	**9.25**
115u574	19,522	16.78	17,625	**9.72**	19,522	0.00	17,679	9.44
115rat575	3199	26.91	2713	15.19	3179	0.63	2693	**15.82**
131p654	33,870	24.64	30,194	10.85	33,870	0.00	29,876	**11.79**
132d657	29,172	28.15	24,421	16.29	29,172	0.00	23,781	**18.48**
145u724	22,465	24.47	19,101	14.97	22,465	0.00	18,891	**15.91**
157rat783	4956	35.15	4068	17.92	–	–	4058	**18.12**
201pr1002	158,910	29.59	141,750	10.80	–	–	141,466	**10.98**
212u1060	150,621	31.43	136,237	9.55	–	–	135,497	**10.04**
217vm1084	173,606	26.38	167,828	3.33	–	–	157,781	**9.12**
MEAN				12.36		2.59		**13.37**
CI				2.14		2.15		2.02

ables for SN-VND are Δ_r and t_r $(r = 1, 2, 3)$ with values chosen in $\{0.05(2H+2), 0.10(2H+2), 0.15(2H+2)\}$ for Δ_1, $\{0.15(2H+2), 0.20(2H+2), 0.25(2H+2)\}$ for Δ_2, $\{0.25(2H+2), 0.30(2H+2), 0.35(2H+2)\}$ for Δ_3. Finally, the time limits for both approaches are chosen in {60, 120, 180, 240, 300, 360} seconds.

The values of the AD-VND design variables identified by the AAC tool are: $\pi_1^* = 1$, $w_1^{1*} = 0$, $w_2^{1*} = 1$, $\alpha_1'^* = 0.10$, $\alpha_1''^* = 0.05$, $\alpha_1'''^* = 0.05$, $t_1^* = 60$ s, $\alpha_2'^* = 0.20$, $\alpha_2''^* = 0.10$, $\alpha_2'''^* = 0.10$, $t_2^* = 60$ s, $\alpha_3'^* = 0.35$, $\alpha_3''^* = 0.175$, $\alpha_3'''^* = 0.175$, and $t_3^* = 120$ s.

The values of the SN-VND design variables after the training phase are: $\Delta_1^* = 0.15(2H+2)$, $t_1^* = 300$ s, $\Delta_2^* = 0.20(2H+2)$, $t_2^* = 300$ s, $\Delta_3^* = 0.30(2H+2)$ and $t_3^* = 300$ s.

The computational results are presented in Tables 3, where the instances are taken from Fischetti et al. [41]. In this case, initial solutions are obtained by choosing a random vertex for each cluster and then by sequencing them with a nearest-neighbor heuristic.

As happens for the TSP, as the instances become larger (instances from 157rat783 on) the SN-VND is not able to load the model (the explanation is the same as for the TSP) and even for smaller instances the improvements in many cases are very small (0% in some cases). Overall, the average SN-VND improvement is about 2.50%, with a maximum value of 9.24% for instance 89pcb442. On the other hand, AD-VND is able to visit solutions whose average objective function value is about 13% better than that of the initial solution, with a minimum value of 4.96% for instance 84fl417 and a maximum value of 18.48% for instance 132d657. In this case, the comparison with ADN shows that the advantage of using neighborhoods of different sizes is not always consistent. Indeed, the results of AD-VND are about 0.5% to 2% better than ADN, with a single case in which ADN is about 0.3% better (instance 115u574) and another case in which the deviation of AD-VND is about 6% higher than ADN (instance 217vm1084). We notice that the differences between the two approaches are further lessened because of the confidence intervals that partially overlap.

4.3. Capacitated Vehicle Routing Problem with Time Windows

The VRPTW can be defined on a directed complete graph $G = (V, A)$, where set V contains $n+2$ vertices. Let H denote the set of vehicles. It is required that each vehicle route starts at vertex 0 and ends at vertex $n+1$ (depots), respectively. The service time at vertex $i \in V$ is denoted by s_i (with $s_0 = s_{n+1} = 0$), whereas each arc $(i, j) \in A$ has associated a travel cost c_{ij} and a travel time t_{ij}. For each vertex i it is defined a time window $[a_i, b_i]$ and a demand q_i, while Q is the vehicles' capacity. The boolean variable x_{ij}^h states if arc (i, j) is traversed by vehicle h. The continuous variable w_i^h represents the start service time for vehicle h at vertex i. Finally, we respectively denote with $\delta^+(i)$ and $\delta^-(j)$ the set of predecessors and the set of successors, i.e. $\delta^+(i) = \{j : (i, j) \in A\}$ and $\delta^-(j) = \{i : (i, j) \in A\}$. We consider the following distance-minimizing formulation [42]:

$$\min z = \sum_{h \in H} \sum_{(i,j) \in A} c_{ij} x_{ij}^h$$

$$\text{s.t.} \sum_{h \in H} \sum_{j \in \delta^+(i)} x_{ij}^h = 1 \qquad i \in N \tag{18}$$

$$\sum_{j \in \delta^+(0)} x_{0j}^h = 1 \qquad h \in H \tag{19}$$

$$\sum_{i \in \delta^-(j)} x_{ij}^h - \sum_{i \in \delta^+(j)} x_{ji}^h = 0 \qquad h \in H, j \in N \tag{20}$$

$$\sum_{i \in \delta^-(n+1)} x_{i,n+1}^h = 1 \qquad h \in H \tag{21}$$

$$x_{ij}^h (w_i^h + s_i + t_{ij} - w_j^h) \leq 0 \qquad h \in H, (i, j) \in A \tag{22}$$

$$\sum_{i \in N} q_i \sum_{j \in \delta^+(i)} x_{ij}^h \leq Q \qquad h \in H \tag{23}$$

$$a_i \leq w_i^h \leq b_i \qquad h \in H, i \in V \tag{24}$$

$$x_{ij}^h \in \{0, 1\} \qquad h \in H, (i, j) \in A. \tag{25}$$

As showed in [42], the above formulation can be linearized by using a big M constant.

In this model, the types of entities are the vertices $(k = 1)$ and the vehicles $(k = 2)$, i.e., $K = 2$. Since the fleet is homogeneous, vehicles do not index any parameter $(p_2 = 0)$. Indeed, the sole vehicles' parameter is the capacity, that is constant. On the other hand, there are six parameters indexed by each vertex entity $(p_1 = 6)$: the vertex-to-vertex costs $(l = 1)$, the service times $(l = 2)$, the release times $(l = 3)$, the deadlines $(l = 4)$, the vertex-to-vertex travel times $(l = 5)$, as well as the demands $(l = 6)$. Hence, the vertices

Table 4
Parameters' values after the training phase for the AD-VND in the VRPTW case. The following parameters have the same values for each class of instances: $\pi_1^* = 0.1, \pi_2^* = 0.9$, $\alpha_1'^* = \alpha_2'^* = \alpha_3'^* = 0.00$, $\alpha_1'''^* = \alpha_2'''^* = \alpha_3'''^* = 0.00$, $w_2^{1*} = 0, w_3^{1*} = 0, w_6^{1*} = 0$.

Dataset	Parameter	No TW	$\beta = 0$	$\beta = 0.25$	$\beta = 0.33$	$\beta = 0.5$
C1	$\alpha_1'^*$	0.12	0.12	0.12	0.06	0.12
	$\alpha_2'^*$	0.21	0.21	0.21	0.21	0.21
	$\alpha_3'^*$	0.24	0.30	0.24	0.30	0.24
	t_1^*	60	60	60	60	60
	t_2^*	60	60	60	60	120
	t_3^*	60	60	60	120	120
	w_1^{1*}	0	1	0	0	1
	w_4^{1*}, w_5^{1*}	1	1	1	0	1
C2	$\alpha_1'^*$	0.06	0.06	0.12	0.12	0.12
	$\alpha_2'^*$	0.15	0.21	0.15	0.21	0.21
	$\alpha_3'^*$	0.24	0.30	0.30	0.30	0.30
	t_1^*, t_2^*, t_3^*	120	60	60	60	120
	w_1^{1*}	0	0	0	0	0
	w_4^{1*}, w_5^{1*}	0	0	0	1	1
R1	$\alpha_1'^*$	0.12	0.12	0.06	0.12	0.06
	$\alpha_2'^*$	0.15	0.21	0.21	0.21	0.21
	$\alpha_3'^*$	0.24	0.30	0.30	0.30	0.30
	t_1^*, t_2^*, t_3^*	60	60	60	120	60
	w_1^{1*}	0	0	0	0	1
	w_4^{1*}, w_5^{1*}	0	1	1	0	0
R2	$\alpha_1'^*$	0.12	0.06	0.06	0.12	0.12
	$\alpha_2'^*$	0.15	0.15	0.21	0.21	0.21
	$\alpha_3'^*$	0.24	0.24	0.30	0.30	0.24
	t_1^*	60	60	60	60	60
	t_2^*	120	60	60	60	120
	t_3^*	120	60	120	60	120
	$w_1^{1*}, w_4^{1*}, w_5^{1*}$	0	0	0	0	0
RC1	$\alpha_1'^*$	0.12	0.12	0.06	0.12	0.12
	$\alpha_2'^*$	0.15	0.15	0.15	0.21	0.21
	$\alpha_3'^*$	0.24	0.30	0.30	0.30	0.30
	t_1^*, t_2^*	60	60	60	60	60
	t_3^*	60	120	120	60	60
	w_1^{1*}	0	0	0	1	0
	w_4^{1*}, w_5^{1*}	0	0	0	1	1
RC2	$\alpha_1'^*$	0.12	0.12	0.06	0.06	0.12
	$\alpha_2'^*$	0.15	0.21	0.15	0.21	0.21
	$\alpha_3'^*$	0.30	0.24	0.30	0.30	0.30
	t_1^*	60	60	60	60	120
	t_2^*	60	60	120	60	120
	t_3^*	120	120	120	120	120
	w_1^{1*}	0	0	0	0	0
	w_4^{1*}, w_5^{1*}	0	0	0	0	1

dataset D_1 has six groups of columns whilst the vehicles dataset D_2 is empty. Therefore, the design variables for the AD-VND are: α_1', α_2', α_3', α_1'', α_2'', α_3'', α_1''', α_2''', α_3''', t_1, t_2, t_3, w_1^1, w_2^1, w_3^1, w_4^1, w_5^1 and w_6^1 as well as π_1 and π_2. The sets of possible values for AD-VND are: {0.06, 0.09, 0.12} for α_1', {0.15, 0.18, 0.21} for α_2', {0.24, 0.27, 0.30} for α_3', {0.000, 0.025, 0.050, 0.075, 0.100, 0.0125, 0.150, 0.175, 0.200} for α_r'' and α_r''' $(r = 1, 2, 3)$, whereas these sets are {0, 1} for w_l^1 $(l = 1, \ldots, 6)$ and {0.1, 0.5, 0.9} for π_1 and π_2. On the other hand, the only design variables for SN-VND are Δ_r and t_r $(r = 1, 2, 3)$ with possible values chosen in $\{0.07\gamma, 0.11\gamma, 0.15\gamma\}$ for Δ_1, $\{0.20\gamma, 0.25\gamma, 0.29\gamma\}$ for Δ_2 and $\{0.37\gamma, 0.44\gamma, 0.51\gamma\}$ for Δ_3, where $\gamma = 2n + |H|$. With respect to the time limits, for both approaches their values are chosen in {60, 120, 180, 240, 300, 360} s.

Our tests have been made on the Solomon's instances [43] which are composed of six classes: C1, C2, R1, R2, RC1, RC2. In particular, class C1 is made up of nine instances, class R1 contains 12 instances, class R2 is composed of 11 instances, whereas classes C2, RC1, and RC2 contain eight instances each. In order to vary the width of the time windows, we have generated additional classes of test instances as follows. Let $[a_i, b_i]$ be the time window of a customer i in a Solomon's instance. We have tightened the time window of each customer i as follows: $[a_i + \beta(b_i - a_i), b_i - \beta(b_i - a_i)]$ for $\beta = 0$, 0.25, 0.33 and 0.5. We have also completely relaxed the time windows by setting $[a_i = a_0, b_i = b_0]$, where $[a_0, b_0]$ is the time window of the vehicle depot.

The values outputted by the AAC tool are reported in Tables 4 and 5 (for AD-VND and SN-VND, respectively), differentiated for each class of instance.

The initial solutions were obtained by using the Constraint Programming model reported in CP Optimizer Forum [44].

The results of our experiments are detailed in Tables 6–10. The AD-VND clearly outperforms the SN-VND. More specifically, the maximum average objective function improvement of the automatically-designed VND is 60.46% for the case when the time windows are relaxed, while the corresponding SN-based VND improvement is 26.20%. This is the case for which the gap between the two VND procedures is the largest. The maximum improvement of SN-VND is 32.68% when $\beta = 0$, still very far from the corresponding AD-VND improvement which turns out to be 60.12%. We observe that the gap between the two VND procedures is almost negligible for $\beta = 0.5$ (about 3%).

When compared with ADN, the AD-VND still performs satisfactorily, even if the improvements are reduced with respect to

Table 5
Parameters' values after the training phase for the SN-VND in the VRPTW case.

Dataset	Parameter	No TW	$\beta = 0$	$\beta = 0.25$	$\beta = 0.33$	$\beta = 0.5$
C1	Δ_1^*	0.15γ	0.15γ	0.15γ	0.15γ	0.15γ
	Δ_2^*	0.29γ	0.29γ	0.29γ	0.20γ	0.29γ
	Δ_3^*	0.51γ	0.51γ	0.51γ	0.37γ	0.37γ
	t_1^*, t_2^*	120	60	60	60	60
	t_3^*	120	60	60	120	60
C2	Δ_1^*	0.15γ	0.15γ	0.15γ	0.15γ	0.07γ
	Δ_2^*	0.29γ	0.29γ	0.29γ	0.20γ	0.29γ
	Δ_3^*	0.37γ	0.37γ	0.51γ	0.51γ	0.51γ
	t_1^*	60	60	60	60	60
	t_2^*	120	60	60	60	120
	t_3^*	120	120	120	60	120
R1	Δ_1^*	0.15γ	0.15γ	0.15γ	0.15γ	0.07γ
	Δ_2^*	0.20γ	0.29γ	0.20γ	0.20γ	0.29γ
	Δ_3^*	0.37γ	0.37γ	0.51γ	0.37γ	0.51γ
	t_1^*, t_2^*	120	120	120	120	60
	t_3^*	120	120	120	120	120
R2	Δ_1^*	0.15γ	0.15γ	0.15γ	0.15γ	0.15γ
	Δ_2^*	0.20γ	0.20γ	0.20γ	0.29γ	0.29γ
	Δ_3^*	0.51γ	0.37γ	0.51γ	0.51γ	0.37γ
	t_1^*	60	60	60	120	60
	t_2^*	120	60	120	120	60
	t_3^*	120	120	120	120	60
RC1	Δ_1^*	0.15γ	0.15γ	0.15γ	0.15γ	0.07γ
	Δ_2^*	0.20γ	0.29γ	0.20γ	0.29γ	0.29γ
	Δ_3^*	0.51γ	0.37γ	0.51γ	0.37γ	0.51γ
	t_1^*	120	60	60	120	60
	t_2^*	120	60	120	120	60
	t_3^*	120	60	120	120	120
RC2	Δ_1^*	0.15γ	0.15γ	0.07γ	0.07γ	0.07γ
	Δ_2^*	0.29γ	0.29γ	0.20γ	0.29γ	0.20γ
	Δ_3^*	0.37γ	0.37γ	0.37γ	0.37γ	0.37γ
	t_1^*	60	120	60	60	120
	t_2^*	60	120	120	60	120
	t_3^*	60	120	120	120	120

the comparison with SN-VND. In particular, the values for DEV_{z_0} are about 1% to 6% greater than ADN. There are only three cases for which ADN is slightly better than our automatically-designed VND. More specifically, this is the case of instances of class C2 and RC2 when the time windows are relaxed (average difference of less than 1%, Table 6), and instances of class R1 when $\beta = 0.25$ (average difference of about 1%, Table 8). As in the GTSP, for some classes of the Solomon's instances the differences between ADN and AD-VND are somehow reduced because of the confidence intervals that partially overlap.

4.4. Multi-Plant, Multi-Item, Multi-Period Capacitated Lot-Sizing Problem

Given a planning horizon partitioned into T periods, the MPCLSP aims to determine a least cost feasible solution for the following decisions: the quantity of items to be produced and then stored at each plant in each period, the quantity of items to be transferred between plants in each period. Let us denote with M and N the set of plants and the set of items to produce, respectively. For each (i, j, t), $i \in N$, $j \in M$, $t \in T$ we define the following parameters: the demand d_{it}; the setup time f_{ijt}; the unit production cost c_{ijt}; the setup cost s_{ijt}; the unit inventory cost h_{ijt}; the unit processing time b_{ijt}. Moreover, we denote with C_{jt} the production capacity of plant j in period t, whereas the parameter $u_{jj't}$ represents the unit transportation cost of an item between plant j and plant j' in period t. For each (i, j, t), $i \in N$, $j \in M$, $t \in T$, we define the decision variables y_{ijt}, x_{ijt} and I_{ijt} as follows. The binary variable y_{ijt} is set to 1 *or* 0 to model the decision to produce *or* not the item i at plant j in period t . The integer variable x_{ijt} represents the quantity of item i manufactured at plant j during period t. The integer variable I_{ijt} denotes the inventory level of item i at plant j at the end of the period t. Finally, the integer variable $q_{ijj't}$ models the decision about the quantity of item i to be transferred from plant j to plant j' in period t. The formulation of the MPCLSP we use is the following [45]:

$$
\begin{aligned}
&\min z = \sum_{i\in N}\sum_{j\in M}\sum_{t\in T}\left(c_{ijt}x_{ijt} + s_{ijt}y_{ijt} + h_{ijt}I_{ijt} + \sum_{j'\in M, j'\neq j} u_{jj't}q_{ijj't}\right)\\
&\text{s.t. } I_{ij,t-1} + x_{ijt} - \sum_{j'\in M, j'\neq j} q_{ijj't} + \sum_{j'\in M, j'\neq j} q_{ij'jt} - I_{ijt} = d_{ijt} \quad i \in N, j \in M, t \in T\\
&x_{ijt} \leq \left(\sum_{j\in M}\sum_{l\in T, l\geq t} d_{ijl}\right) y_{ijt} \qquad i \in N, j \in M, t \in T\\
&\sum_{i\in N}\left(b_{ijt}x_{ijt} + f_{ijt}y_{ijt}\right) \leq C_{jt} \qquad j \in M, t \in T\\
&I_{ij0} = 0 \qquad i \in N, j \in M\\
&x_{ijt}, I_{ijt} \geq 0 \qquad i \in N, j \in M, t \in T\\
&q_{ijj't} \geq 0 \qquad i \in N, j \in M, j' \in M, t \in T\\
&y_{ijt} \in \{0, 1\} \qquad i \in N, j \in M, t \in T.
\end{aligned} \tag{26}
$$

In this formulation, we have three types ($K = 3$) of entities: the items ($k = 1$), the plants ($k = 2$) and the time periods ($k = 3$). Each item indexes five parameters ($p_1 = 5$), while there are seven parameters indexed by plants and time periods ($p_2 = p_3 = 7$). As described in Section 3.1, we define three datasets: an items dataset D_1 of $|N|$ rows and five groups of columns, a plants dataset D_2 of $|M|$ rows and seven groups of columns and a time periods dataset D_3 of T rows and seven groups of columns. Therefore, the design variables for AD-VND are: α'_r, α''_r, α'''_r, t_r $(r = 1, 2, 3)$, π_1, π_2, π_3, w_l^1 $(l = 1, \ldots, 5)$, w_l^2 $(l = 1, \ldots, 7)$ and w_l^3 $(l = 1, \ldots, 7)$. In particular, the sets of possible values that we consider are {0.005, 0.010, 0.015} for α'_1, {0.015, 0.020, 0.025} for α'_2, {0.025, 0.030, 0.035} for α'_3, and {0.000, 0.025, 0.050, 0.075, 0.100, 0.0125, 0.150, 0.175, 0.200} for α''_r and α'''_r $(r = 1, 2, 3)$. Moreover, the values of π_1, π_2, π_3 range in {0.00, 0.33, 0.50, 1.00}, whereas every w is chosen in {0, 1}. With respect to SN-VND, the possible values for Δ_r $(r = 1, 2, 3)$ are selected in $\{\beta/512, \beta/256, \beta/128\}$ for Δ_1, $\{\beta/128, \beta/64, \beta/32\}$ for Δ_2 and $\{\beta/32, \beta/16, \beta/8\}$ for Δ_3, where $\beta = |N| \cdot |M| \cdot T$.

Table 6
Results for the VRPTW when the time windows are relaxed. The values z_0, GAP, z, and DEV_{z_0} are averaged over the several instances composing each class.

Instance	z_0	GAP	ADN			SN-VND			AD-VND		
		(%)	z	DEV_{z_0} (%)	CI (%)	z	DEV_{z_0} (%)	CI (%)	z	DEV_{z_0} (%)	CI (%)
C1	4,095,069.33	92.38	1,753,815.93	57.23	1.31	2,884,524.33	28.66	3.41	1,540,821.00	**62.12**	1.61
C2	3,468,978.75	83.51	1,135,744.72	**64.30**	2.20	2,224,204.63	34.34	3.36	1,146,191.88	63.74	1.44
R1	3,183,083.50	81.47	1,543,985.46	50.50	4.85	2,610,826.50	18.10	4.00	1,527,623.92	**50.96**	3.07
R2	3,765,233.27	84.50	1,282,060.10	65.74	2.31	2,700,543.73	28.25	4.37	1,240,245.73	**66.91**	2.31
RC1	3,825,735.63	85.90	1,808,967.64	51.42	4.51	3,103,693.25	18.59	2.19	1,804,874.50	**51.68**	4.84
RC2	4,793,855.75	88.91	1,508,102.78	**68.40**	2.74	3,382,901.38	29.22	3.30	1,558,726.50	67.33	2.73
MEAN				59.60			26.20			**60.46**	

Table 7
Results for the VRPTW with $\beta = 0$. The values z_0, GAP, z, and DEV_{z_0} are averaged over the several instances composing each class.

Instance	z_0	GAP	ADN			SN-VND			AD-VND		
		(%)	z	DEV_{z_0} (%)	CI (%)	z	DEV_{z_0} (%)	CI (%)	z	DEV_{z_0} (%)	CI (%)
C1	4,095,069.33	85.66	1,489,235.92	63.47	1.36	2,488,458.89	36.86	4.54	1,450,537.00	**63.95**	4.31
C2	3,468,978.75	81.83	1,153,256.08	64.18	4.85	1,977,009.63	45.56	4.50	958,137.13	**69.37**	4.49
R1	3,183,083.50	74.07	1,708,897.03	44.89	4.82	2,327,596.50	26.82	2.97	1,687,489.50	**45.99**	5.09
R2	3,765,233.27	81.98	1,315,912.04	64.78	2.31	2,602,982.64	30.97	3.83	1,289,774.18	**65.64**	1.89
RC1	3,825,735.63	81.30	2,010,863.24	45.89	4.77	2,913,944.63	23.44	4.25	2,029,636.75	**45.95**	4.06
RC2	4,793,855.75	86.21	1,563,805.27	67.20	2.32	3,229,064.75	32.46	3.55	1,440,445.13	**69.83**	2.13
MEAN				58.40			32.68			**60.12**	

Table 8
Results for the VRPTW with $\beta = 0.25$. The values z_0, GAP, z, and DEV_{z_0} are averaged over the several instances composing each class.

Instance	z_0	GAP	ADN			SN-VND			AD-VND		
		(%)	z	DEV_{z_0} (%)	CI (%)	z	DEV_{z_0} (%)	CI (%)	z	DEV_{z_0} (%)	CI (%)
C1	3,908,173.00	77.87	1,614,729.37	58.45	3.45	2,282,293.78	39.75	4.68	1,465,473.89	**60.84**	4.33
C2	3,754,977.13	81.77	1,146,491.76	65.85	1.40	2,173,373.63	42.43	4.24	982,430.88	**71.07**	4.79
R1	2,545,156.67	58.10	1,902,210.99	**24.44**	4.58	1,953,402.33	22.78	3.57	1,918,718.08	23.42	3.53
R2	3,252,372.73	74.66	1,430,847.67	55.49	3.52	2,298,351.09	28.92	3.74	1,317,565.45	**58.86**	4.54
RC1	2,868,663.50	61.40	2,358,259.05	17.60	2.04	2,309,612.75	19.36	3.06	2,061,248.00	**27.94**	3.65
RC2	4,444,461.13	79.90	1,705,719.05	60.97	4.95	2,925,159.75	33.86	2.70	1,588,804.25	**63.40**	4.18
MEAN				47.13			31.18			**50.92**	

Table 9
Results for the VRPTW with $\beta = 0.33$. The values z_0, GAP, z, and DEV_{z_0} are averaged over the several instances composing each class.

Instance	z_0	GAP	ADN			SN-VND			AD-VND		
		(%)	z	DEV_{z_0} (%)	CI (%)	z	DEV_{z_0} (%)	CI (%)	z	DEV_{z_0} (%)	CI (%)
C1	3,939,832.22	73.35	1,716,724.92	55.96	5.57	2,184,093.00	42.84	4.03	1,525,932.89	**60.13**	5.39
C2	4,185,462.63	84.83	1,218,299.86	70.08	4.44	2,282,243.00	44.83	4.88	1,073,760.13	**73.27**	5.66
R1	2,438,978.33	51.30	2,004,816.67	17.15	2.63	1,917,594.83	20.86	2.11	1,823,316.92	**24.73**	2.50
R2	3,230,956.00	70.70	1,491,276.46	53.31	3.80	2,197,847.36	31.78	2.98	1,287,632.91	**59.70**	3.47
RC1	2,896,091.57	50.85	2,456,283.89	14.68	3.23	2,352,513.57	18.63	3.44	2,191,363.29	**23.83**	3.40
RC2	4,455,520.63	76.95	1,720,859.19	60.99	4.77	2,920,403.88	34.31	2.88	1,535,787.25	**65.22**	3.34
MEAN				45.36			32.21			**51.15**	

Table 10
Results for the VRPTW with $\beta = 0.5$. The values z_0, GAP, z, and DEV_{z_0} are averaged over the several instances composing each class.

Instance	z_0	GAP	ADN			SN-VND			AD-VND		
		(%)	z	DEV_{z_0} (%)	CI (%)	z	DEV_{z_0} (%)	CI (%)	z	DEV_{z_0} (%)	CI (%)
C1	4,077,181.89	30.92	2,636,402.20	25.01	3.10	2,471,204.78	22.96	3.97	2,206,442.33	**27.05**	4.66
C2	4,178,275.63	39.29	1,916,409.35	38.06	4.73	2,327,125.88	31.15	3.90	1,843,218.00	**39.29**	4.88
R1	3,905,566.83	21.16	3,000,584.19	18.14	1.91	2,887,368.42	20.41	1.97	2,850,071.25	**21.16**	2.06
R2	3,659,437.09	28.82	2,351,581.43	26.21	2.81	2,365,718.45	25.93	2.63	2,221,423.27	**28.82**	3.00
RC1	5,212,424.13	23.87	3,835,937.88	20.80	2.79	3,705,231.38	22.85	2.74	3,637,545.00	**23.87**	2.92
RC2	3,516,686.88	14.83	2,608,179.43	13.73	3.08	2,715,935.88	12.10	2.60	2,535,352.50	**14.83**	3.30
MEAN				23.66			22.57			**25.84**	

Finally, the time limits for both approaches are chosen in {60, 120, 180, 240, 300, 360, 420, 480, 540, 600} s.

The values of the design variables for the AD-VND are: $\pi_1^* = 1$, $\pi_2^* = 0$, $\pi_3^* = 0$, $w_l^{1*} = 0$ $(l = 1, \ldots, 5)$, $w_l^{2*} = 0$ $(l = 1, \ldots, 7)$, $w_l^{3*} = 0$ $(l = 1, \ldots 7)$, $\alpha_1'^* = 0.010$, $\alpha_2'^* = 0.015$, $\alpha_3'^* = 0.025$, $t_1^* = 120$ s, $t_2^* = 240$ s, $t_3^* = 360$ s, $\alpha_r''^* = 0.00$ $(r = 1, 2, 3)$ and $\alpha_r'''^* = 0.00$ $(r = 1, 2, 3)$. As in the TSP, the meaning of all weights w and the parameters α_r'' and α_r''' $(r = 1, 2, 3)$ being zero is that the sets F_r $(r = 1, 2, 3)$ coincide with sets F_r', i.e., the neighborhood structures are directly influenced only by the solution-based features.

On the other hand, the values of the design variables for the SN-VND are: $\Delta_1^* = \beta/128$, $\Delta_2^* = \beta/64$, $\Delta_3^* = \beta/32$. With respect to the time limits, the AAC procedure has produced different values depending on the set of instances. In particular, for the instances with 80 items the values are $t_1^* = 120$ s, $t_2^* = 240$ s, $t_3^* = 360$ s, while for the instances with 150 items the values are $t_1^* = 480$ s, $t_2^* = 540$ s, $t_3^* = 600$ s.

Experiments for this problem have been performed by using four classes of 10 instances randomly generated according to Nascimento et al. [45], with 12 time periods, 15, 20 or 30 plants, and the item set containing either 80 or 150 elements. The initial solutions are obtained by running the MIP solver and stopping it after finding the first feasible solution.

The results are reported in Table 11. For this problem, the AD-VND is slightly better than the SN-VND (3.84% compared to 2.17% on the average). Moreover, in one case (12 time periods, 30 plants

Table 11
Results for the MPCLSP. The values z_0, GAP, z, and DEV_{z_0} are averaged over the several instances composing each class.

Instance			z_0	GAP	ADN			SN-VND			AD-VND		
T	$\|M\|$	$\|N\|$		(%)	z	DEV_{z_0} (%)	CI (%)	z	DEV_{z_0} (%)	CI (%)	z	DEV_{z_0} (%)	CI (%)
12	15	80	2,732,591.28	5.62	2,651,807.16	2.96	0.08	2,651,680.89	2.96	0.67	2,646,774.87	**3.14**	0.09
	20	80	3,814,898.44	9.95	3,594,061.98	5.52	2.86	3,611,242.19	5.13	2.36	3,558,121.36	**6.47**	2.82
		150	6,767,476.22	5.35	6,608,230.43	2.35	0.10	6,726,335.84	0.61	0.15	6,542,148.13	**3.33**	0.10
	30	150	10,036,839.70	5.10	9,890,667.62	1.46	0.06	10,036,839.70	0.00	–	9,791,760.94	**2.44**	0.06
MEAN						3.07			2.17			**3.84**	

Table 12
Analysis of the performance of the two model-based VND procedures.

Test problem	SN-VND				AD-VND			
	Visited solutions			Local Optima	Visited solutions			Local Optima
	$r=1$	$r=2$	$r=3$	(%)	$r=1$	$r=2$	$r=3$	(%)
TSP	2.00	2.94	1.00	87.68	5.66	6.00	5.83	95.93
GTSP	2.77	1.17	1.00	80.56	5.70	3.49	1.45	94.57
VRPTW No TW	2.36	1.60	1.12	84.32	2.60	3.08	3.23	98.63
VRPTW $\beta=0$	3.19	1.43	1.00	90.28	1.94	2.57	2.98	98.51
VRPTW $\beta=0.25$	3.36	1.75	1.00	91.10	1.55	2.35	2.93	98.09
VRPTW $\beta=0.33$	4.28	2.00	–	92.22	1.72	2.19	3.32	99.02
VRPTW $\beta=0.5$	1.44	1.03	1.00	94.98	1.34	1.34	1.16	100.00
MPCLSP	2.10	2.01	1.30	78.22	1.39	1.73	3.47	100.00

and 150 items) SN-VND does not improve the initial solution (the improvement of AD-VND is 2.44%) and in another case (12 time periods, 20 plants and 150 items) the improvement is limited to 0.61% (compared to 3.33% of AD-VND).

With respect to ADN, we observe that our automatically-designed VND consistently achieve better results, even if the improvements are not considerably higher. The minimum difference between the two approaches is about 0.20%, whereas the maximum value is about 1%. In this case, we notice that the confidence intervals overlap only for the class of instances with 12 periods, 20 plants and 80 items.

4.5. Further analysis

In this subsection we further analyze the performance of the two model-based VND procedures in terms of how precisely they search the different neighborhoods, as well as of the number of solutions visited for each neighborhood. In particular, in Table 12 we report, for each test problem and for each neighborhood structure in the hierarchy, the average number of solutions explored by the two approaches, as well as the percentage of local optima found during the whole search (computed as the number of local optima found divided by the overall number of neighborhoods explored).

In general, we observe that AD-VND explores more solutions than SN-VND, with a few exceptions that typically are limited to $r=1$. The difference in the number of neighbors visited becomes more evident as the size of the neighborhood structures increases. We also notice that for the VRPTW with $\beta=0.33$ SN-VND never makes use of the biggest neighborhood structure (a ' - ' is reported in the corresponding entry). With respect to the percentage of local optima certified during the whole search (i.e., when the exploration of a neighborhood has been performed exhaustively before the time limit), the results show that both approaches find a considerable number of local optima. However, AD-VND is able to exhaustively explore the different neighborhoods more often than SN-VND for each test problem (about 87% for SN-VND compared to about 98% for AD-VND).

5. Conclusions

This paper has proposed a procedure to automatically instantiate a Variable Neighborhood Descent procedure from a MIP model. In particular, we have moved on from a recent paper in which a single neighborhood structure is automatically designed from a MIP model. Here, we have generalized this approach and defined a procedure that - based on a MIP model and a current feasible solution - defines automatically the hierarchy of neighborhood structures of an entire VND algorithm. Our computational results have tested the performance of our approach on four well-known combinatorial optimization problems. In particular, our automatically-generated VND has been compared with the previous single-neighborhood procedure and another model-derived VND procedure. In the first case, the results have confirmed that using a hierarchy of neighborhood structures allow to achieve better solutions, even if in some cases the improvement is not very high. On the other hand, the second comparison shows that our automatically-generated VND outperforms the other model-derived VND procedure with respect to both scalability and solution quality.

Acknowledgments

This work was partly supported by the Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR) of Italy (PON project PON01_00878 "DIRECT FOOD"). This support is gratefully acknowledged. The authors also thank three anonymous referees for their useful comments, which helped to improve the paper.

References

[1] Hinkelmann K, Kempthorne O. Design and analysis of experiments, special designs and applications, vol. 3. John Wiley & Sons; 2012.
[2] Koza JR. Human-competitive results produced by genetic programming. Genet Program Evol M 2010;11:251–84.
[3] Adamo T, Ghiani G, Grieco A, Guerriero E, Manni E. MIP neighborhood synthesis through semantic feature extraction and automatic algorithm configuration. Comput Oper Res 2017;83:106–19.
[4] Hansen P, Mladenović N, Brimberg J, Moreno Pérez JA. Variable neighborhood search. In: Gendreau M, Potvin J-Y, editors. Handbook of metaheuristics. International series in operations research & management science. Springer; 2010. p. 61–86.

[5] Fischetti M, Lodi A. Local branching. Math Prog Ser-B 2003;98(1–3):23–47.
[6] Danna E, Rothberg E, Le Pape C. Exploring relaxation induced neighborhoods to improve MIP solutions. Math Prog Ser-A 2005;102:71–90.
[7] Parisini F, Milano M. Sliced neighborhood search. Expert Syst Appl 2012;39:5739–47.
[8] Ghiani G, Laporte G, Manni E. Model-based automatic neighborhood design by unsupervised learning. Comput Oper Res 2015;54:108–16.
[9] Adamo T, Calogiuri T, Ghiani G, Grieco A, Guerriero E, Manni E. Neighborhood synthesis from an ensemble of MIP and CP models. In: Festa P, Sellmann M, Vanschoren J, editors. Learning and intelligent optimization: 10th international conference, LION 10, Ischia, Italy, May 29, – June 1, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10079. Cham: Springer International Publishing; 2016. p. 221–6.
[10] Maniezzo V, Stützle T, Voß S, editors. Matheuristics: hybridizing metaheuristics and mathematical programming, Vol. 10. Annals of Information Systems. Heidelberg: Springer; 2009.
[11] Hansen P, Mladenović N, Urošević D. Variable neighborhood search and local branching. Comput Oper Res 2006;33(10):3034–45.
[12] Lazić J, Hanafi S, Mladenović N, Urošević D. Variable neighbourhood decomposition search for 0–1 mixed integer programs. Comput Oper Res 2010;37(6):1055–67.
[13] Pillac V, Van Hentenryck P, Even C. A path-generation matheuristic for large scale evacuation planning. Cham: Springer International Publishing; 2014. p. 71–84.
[14] Blum C, Pinacho P, López-Ibáñez M, Lozano J. Construct, merge, solve & adapt a new general algorithm for combinatorial optimization. Comput Oper Res 2016;68:75–88.
[15] Van Hentenryck P, Michel L. Synthesis of constraint-based local search algorithms from high-level models. In: Proceedings of the national conference on artificial intelligence, vol. 22; 2007. p. 273–8.
[16] Elsayed SAM, Michel L. Synthesis of search algorithms from high-level CP models. In: Lee J, editor. Principles and practice of constraint programming – CP 2011. Lecture Notes in Computer Science, vol. 6876. Springer Berlin Heidelberg; 2011. p. 256–70.
[17] Van Hentenryck P, Michel L. Constraint-based local search. The MIT Press; 2009.
[18] Mouthuy S, Van Hentenryck P, Deville Y. Constraint-based very large-scale neighborhood search. Constraints 2012;87:87–122.
[19] Kiziltan Z, Lodi A, Milano M, Parisini F. Bounding, filtering and diversification in CP-based local branching. J Heuristics 2012;18:353–74.
[20] Burke EK, Hyde M, Kendall G, Ochoa G, Özcan E, Woodward JR. A classification of hyper-heuristic approaches. In: Handbook of metaheuristics. Springer; 2010. p. 449–68.
[21] Fukunaga AS. Automated discovery of local search heuristics for satisfiability testing. Evol Comput 2008;16(1):31–61.
[22] Burke EK, Hyde MR, Kendall G, Ochoa G, Özcan E, Woodward JR. Exploring hyper-heuristic methodologies with genetic programming. In: Mumford CL, Jain LC, editors. Computational intelligence: collaboration, fusion and emergence. Berlin Heidelberg: Springer Verlag; 2009. p. 177–201.
[23] van Lon RRS, Holvoet T, Vanden Berghe G, Wenseleers T, Branke J. Evolutionary synthesis of multi-agent systems for dynamic dial-a-ride problems. In: Proceedings of the 14th annual conference companion on genetic and evolutionary computation. GECCO '12. New York, NY, USA: ACM; 2012. p. 331–6.
[24] Hutter F, Hoos HH, Stützle T. Automatic algorithm configuration based on local search. In: Proceedings of the nationalyy conference on artificial intelligence, vol. 22; 2007. p. 1152–7.
[25] Birattari M. Tuning metaheuristics: a machine learning perspective. Springer Publishing Company, Inc.; 2009.
[26] Hoos HH. Programming by optimization. Commun ACM 2012;55:70–80.
[27] Birattari M, Stützle T, Paquete L, Varrentrapp K. A racing algorithm for configuring metaheuristics. In: Proceedings of the genetic and evolutionary computation conference. GECCO '02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2002. p. 11–18.
[28] Balaprakash P, Birattari M, Stützle T. Improvement strategies for the F-Race algorithm: sampling design and iterative refinement. In: Bartz-Beielstein T, editor. Hybrid metaheuristics. Lecture Notes in Computer Science, vol. 4771. Springer Berlin Heidelberg; 2007. p. 108–22.
[29] Adenso-Díaz B, Laguna M. Fine-tuning of algorithms using fractional experimental designs and local search. Oper Res 2006;54(1):99–114.
[30] Hutter F, Hoos HH, Leyton-Brown K, Stützle T. Paramils: an automatic algorithm configuration framework.. J Artif Intell Res 2009;36:267–306.
[31] López-Ibánez M, Dubois-Lacoste J, Pérez Cáceres L, Birattari M, Stützle T. The irace package: iterated racing for automatic algorithm configuration. Oper Res Perspect 2016;3:43–58.
[32] Mascia F, López-Ibáñez M, Dubois-Lacoste J, Stützle T. Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. Comput Oper Res 2014;51:190–9.
[33] Russell S, Norvig P. Artificial intelligence: a modern approach. 3rd. Prentice-Hall; 2010.
[34] Fourer R, Gay DM, Kernighan BW. A modeling language for mathematical programming. Manage Sci 1990;36:519–54.
[35] Rosenthal RE. GAMS - a user'S guide. Washington, DC, USA: GAMS Development Corporation; 2017.
[36] Van Hentenryck P. The OPL optimization programming language. MIT Press, Cambridge; 1999.
[37] Batini C, Ceri S, Navathe S. Conceptual database design - an entity-relationship approach. Redwood City, CA, USA: The Benjamin/Cummings Publishing Company, Inc.; 1992.
[38] Miller CE, Tucker AW, Zemlin RA. Integer programming formulations and traveling salesman problems. J ACM 1960;7(4):326–9.
[39] Parr T. The definitive ANTLR 4 reference. 2nd. Pragmatic Bookshelf; 2013.
[40] Reinelt G. TSPLIB - a traveling salesman problem library. ORSA J Comput 1991;3(4):376–84.
[41] Fischetti M, Salazar-González JJ, Toth P. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. Oper Res 1997;45(3):378–94.
[42] Desrochers M, Lenstra JK, Savelsbergh MWP, Soumis F. Vehicle routing with time windows: optimization and approximation. In: Golden BL, Assad AA, editors. Vehicle routing: methods and studies. Amsterdam: North-Holland; 1988. p. 65–84.
[43] Solomon MM. Algorithms for the vehicle routing and scheduling problems with time window constraints. Oper Res 1987;35:254–65.
[44] CP Optimizer Forum. 2017. Last accessed on March, 23rd 2017; URL https://www.ibm.com/developerworks/community/forums/ajax/download/77777777-0000-0000-0000-000014932402/b9379ac1-721e-4d9f-aa4f-e768de6502b4/attachment_14932402_CVRPTW.mod.
[45] Nascimento MCV, Resende MGC, Toledo FMB. GRASP heuristic with path-relinking for the multi-plant capacitated lot sizing problem. Eur J Oper Res 2010;200(3):747–54.