Variable Neighborhood Descent

12

Abraham Duarte, Nenad Mladenović, Jesús Sánchez-Oro, and Raca Todosijević

Contents

A. Duarte (✉)
Department of Ciencias de la Computación, Universidad Rey Juan Carlos, Móstoles (Madrid), Madrid, Spain
e-mail: abraham.duarte@urjc.es

J. Sánchez-Oro
Universidad Rey Juan Carlos, Móstoles (Madrid), Madrid, Spain
e-mail: jesus.sanchezoro@urjc.es

N. Mladenović
GERAD and Ecole des Hautes Etudes Commerciales, Montréal, QC, Canada

LAMIH, University of Valenciennes, Famars, France

LAMIH, France and Mathematical Institute, SANU, Université de Valenciennes, Belgrade, Serbia
e-mail: nenadmladenovic12@gmail.com; Nenad.Mladenovic@univ-valenciennes.fr

R. Todosijević
LAMIH, France and Mathematical Institute, SANU, Université de Valenciennes, Belgrade, Serbia
e-mail: racatodosijevic@gmail.com

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_9

Abstract

Local search heuristic that explores several neighborhood structures in a deterministic way is called variable neighborhood descent (VND). Its success is based on the simple fact that different neighborhood structures do not usually have the same local minimum. Thus, the local optima trap problem may be resolved by deterministic change of neighborhoods. VND may be seen as a local search routine and therefore could be used within other metaheuristics. In this chapter, we discuss typical problems that arise in developing VND heuristic: what neighborhood structures could be used, what would be their order, what rule of their change during the search would be used, etc. Comparative analysis of usual sequential VND variants is performed in solving traveling salesman problem.

Keywords

Variable neighborhood descent · Local search · Intensification · Deterministic exploration

Introduction

Optimization is a key discipline in fields such as computer science, artificial intelligence, and operations research. Outside these scientific communities, the meaning of optimization becomes quite vague, going to mean simply "do it as better as you can." In the context of this chapter, the concept of optimization is conceived as the process of trying to find the best possible solution to an optimization problem, usually in a limited time horizon.

In the simplest case, an optimization problem may be defined by a 2-tuple (X, f), where X represents the set of feasible solutions and f is an objective function that assigns a real number to each solution $x \in X$, which represents its quality or fitness. Then, the main objective of an optimization problem is to find a solution $x^\star \in X$ with the best objective function value among all solutions in the search space. Therefore, in a minimization problem, $x^\star \in X$ is a minimum point if $f(x^\star) \leq f(x)$, $\forall x \in X$. Notice that minimization of f is equivalent to maximization of $-f$.

In optimization problems, there is usually either finite but a huge number or infinity number of solutions and a clear criterion for the comparison among them. Some well-known examples of optimization problems are the traveling salesman problem (TSP), the vehicle routing problem (VRP), the quadratic assignment problems (QAP), or scheduling problems, among others. A detailed description of these problems can be found in [3, 36]. The difficulty of solving these problems has been studied since the late 1970s [12]. These studies concluded that there is a subset of problems where it is possible to design an algorithm, which presents a polynomial computational complexity, i.e., the execution time of these algorithms polynomially grows with the problem size. Such problems belong to the class $\mathcal{P}$, and they are considered "easy to solve." Examples of these problems are the shortest path problem (Dijkstra algorithm), the minimum spanning tree (Prim or Kruskal algorithms), or flows in networks (Ford-Fulkerson algorithm). However, computing optimal solutions is intractable for many optimization problems of industrial and scientific importance (i.e., there is no known algorithm with polynomial complexity to solve it optimally). This type of problems belongs to a class known as $\mathcal{NP}$, and they are considered “hard to solve.”

In practice, we are usually satisfied with “good” solutions, which are obtained by heuristic algorithms. In particular, metaheuristics (MHs) represent a family of approximate [42] optimization techniques that gained a lot of popularity in the past two decades, becoming the most promising and successful techniques for solving hard problems. Unlike exact optimization algorithms, metaheuristics do not guarantee the optimality of the obtained solutions. Additionally, metaheuristics do not define how close the obtained solutions are from the optimal ones, in contrast with approximation algorithms. MHs provide acceptable solutions in a reasonable computing time for solving hard and complex problems in science and engineering.

The term metaheuristic was coined in 1986 [13] as a way of defining *a master process that guides and modifies other subordinate heuristics to explore solutions beyond simple local optimality*. MHs are among the most prominent and successful techniques to solve a large amount of complex and computationally hard combinatorial and numerical optimization problems arising in human activities, such as economics, industry, or engineering. MHs can be seen as general algorithmic frameworks that require relatively few modifications to be adapted to tackle a specific problem. They constitute a very diverse family of optimization algorithms including methods such as simulated annealing (SA), Tabu search (TS), genetic algorithms (GA), ant colony optimization (ACO), or variable neighborhood search (VNS).

Metaheuristics are high-level strategies for exploring the search space using different methods. The search strategies are highly dependent on the philosophy of the metaheuristic itself. In particular, *trajectory-based metaheuristics* can be seen as intelligent extensions of traditional local search methods. The goal of this kind of MH is to escape from a local optimum in order to proceed in the exploration of the search space and move on to find other hopefully better local optimum. Examples of these MHs are Tabu search [13], simulated annealing [23], or variable neighborhood search [15], among others. *Population-based metaheuristics* deal with a set of solutions instead of dealing with only one solution. These techniques proved a natural and intrinsic way for the exploration of the search space. The final performance of these methods strongly depends on how the population is managed. Examples of population-based metaheuristics are genetic algorithms [19], scatter search [14], or memetic algorithms [33], among others. Some authors consider a third kind of metaheuristics called *constructive-based metaheuristics*, where the main effort is put in the intelligent construction of the solution. In other words, instead of starting the search from a random solution, these methods try to construct a high-quality initial solution. Examples of constructive-based metaheuristics are GRASP [10], ant colony optimization [5], or iterated greedy [39], among others.

The number of new proposed metaheuristics has amazingly increased in the last 25 years. Nowadays, the portfolio of MHs contains more than 50 variants, only considering the most stabilized ones (MHs successfully applied to a relatively large set of optimization problems). However, at the end, when designing a metaheuristic for an optimization problem, we face with two contradictory criteria: intensification (exploitation) and diversification (exploration). In fact, the performance of a metaheuristic basically relies on how it balances both criteria. The intensification of an algorithm describes its ability to thoroughly explore the promising regions in the hope to find better solutions. On the other hand, diversification describes the ability of the metaheuristic to explore non-visited regions of the search space in order to assure the evenly exploration of the search space and to avoid the confinement to the procedure to a reduced number of regions. Therefore, when tackling an optimization problem, it is necessary to search for the equilibrium between both criteria.

Variable neighborhood search (VNS) is a metaheuristic which was proposed in [32] as a general framework to solve hard optimization problems. This methodology is based on performing systematic changes of neighborhoods during the search space exploration. VNS has evolved in recent years, resulting in a large variety of strategies. Some of the most relevant variants are reduced VNS (RVNS), variable neighborhood descent (VND), basic VNS (BVNS), skewed VNS (SVNS), general VNS (GVNS), or variable neighborhood decomposition search (VNDS), among others (see [18] for a survey on VNS). We refer the reader to [6, 8, 37, 40, 41] to some recent and successful applications of VNS to hard optimization problems.

In this chapter, we focus on the deterministic variant of VNS, namely, variable neighborhood descent (VND). VND deserves separate attention since it is usually used in context of other metaheuristics as a local search routine. Once the set of neighborhood structures is selected, to be used in a deterministic manner, the VND-based local searches may be designed in three possible ways: (i) sequential VND, where neighborhoods are placed in the list with a given order and always explored in that order; (ii) nested or composite VND, where neighborhood operators are composed, i.e., $N_1(N_2(N_3(\ldots(x))))$ (neighborhood one of neighborhood two of neighborhood three, etc. of x; and (iii) mixed nested VND, where the two previous strategies are combined.

In this chapter, we first give some possible classification of neighborhood structures that are usually used in solving continuous and discrete optimization problems. Then we provide pseudocodes of sequential, composite, and mixed nested variants of VND. Variants of sequential VND are compared on traveling salesman problem.

Neighborhoods

The representation (encoding) of a solution in an optimization problem plays a relevant topic in the design of an algorithm. In fact, as it is well documented, this representation determines the difficulty of solving a problem [12] and also the complexity of some routines within algorithm. In addition, this encoding strongly influences the way in which the neighborhoods of a given solution are defined. Therefore, it is not possible to separate the neighborhood of a solution from its corresponding representation in the computer memory.

Let us assume without loss of generality that each solution $x \in X$ is represented by a vector $x = (x_1, \ldots, x_n)$, being n the size of the problem. Depending on the values of each x_i, we can distinguish among different types of problems: continuous ($x_i \in \mathbb{R}$), binary ($x_i \in \{0, 1\}$), integer ($x_i \in \mathbb{N}$), or permutations ($x_i \in \mathbb{N}$, $1 \leq x_i \leq n$ and $x_i \neq x_j$ if $i \neq j$). We could also identify a fifth class of problems which encompasses mixed variables. In this section, we describe the most common neighborhoods defined in the related literature as well as some basic properties.

Neighborhoods for Continuous Optimization Problems

The continuous constrained nonlinear global optimization problem (GOP) in general form is given as follows:

$$(GOP) \quad \left[\begin{array}{lll} \min & f(x) & \\ \text{s.t.} & g_i(x) \leq 0 & \forall i \in \{1, 2, \ldots, m\} \\ & h_i(x) = 0 & \forall i \in \{1, 2, \ldots, r\} \\ & a_j \leq x_j \leq b_j & \forall j \in \{1, 2, \ldots, n\} \end{array} \right.$$

where $x \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $g_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, 2, \ldots, m$, and $h_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, 2, \ldots, r$ are possibly nonlinear continuous functions and $a, b \in \mathbb{R}^n$ are the variable bounds.

GOP naturally arises in many applications, e.g., in advanced engineering design, data analysis, financial planning, risk management, scientific modeling, etc. Most cases of practical interest are characterized by multiple local optima, and, therefore, in order to find the globally optimal solution, a global scope search effort is needed.

If the feasible set X is convex and objective function f is convex, then GOP is relatively easy to solve, i.e., the Karush-Kuhn-Tucker conditions may be applied. However, if X is not a convex set or f is not a convex function, we could have many local minima, and thus, the problem may not be solved by using classical techniques.

For solving GOP, neighborhood structures $\mathcal{N}_k(x)$ are usually induced from the ℓ_p metric:

$$\rho(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p} \quad (1 \leq p < \infty) \tag{1}$$

or

$$\rho(x, y) = \max_{1 \leq i \leq n} |x_i - y_i| \quad (p \to \infty). \tag{2}$$

The neighborhood $\mathcal{N}_k(x)$ denotes the set of solutions in the k–th neighborhood of x, and using the metric ρ, it is defined as:

$$\mathcal{N}_k(x) = \{y \in X \mid \rho(x, y) \leq \rho_k\}, \tag{3}$$

or

$$\mathcal{N}_k(x) = \{y \in X \mid \rho_{k-1} \leq \rho(x, y) \leq \rho_k\}, \tag{4}$$

where ρ_k, known as the radius of $\mathcal{N}_k(x)$, is monotonically increasing with k.

Note that in some papers neighborhoods structures in $\mathbb{R}^n$ are not induced from the ℓ_p metric (see e.g., [1]).

Neighborhoods for Binary Problems

There are a large family of optimization problems, whose solution is usually represented as a binary array, where the presence or absence of an element is described by means of a binary variable. The knapsack problem [28], the max-cut problem (MCP) [29], or the maximum diversity problem [7] are examples of these problems. For solving binary problems, neighborhood structures $\mathcal{N}_k(x)$ are usually induced from the Hamming metric:

$$d_H(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{5}$$

More precisely, the k–th neighborhood of a solution x, i.e., $\mathcal{N}_k(x)$, relatively to Hamming metric, is defined as

$$\mathcal{N}_k^{Bin}(x) = \{y \in X \mid d_H(x, y) = k\} \tag{6}$$

We will use the max-cut problem to illustrate the most relevant characteristics of binary problems. Consider a graph $G = (V, E)$ with vertex set V and edge set E. Let w_{ij} be the weight associated with edge $(i, j) \in E$. A $cut(W, W')$ is a partition of V into two sets W, $W' = V \setminus W$, and its value is given by the expression:

$$cut(W, W') = \sum_{i \in W \wedge j \in W'} w_{ij}$$

The max-cut problem (MCP) consists of finding a cut in G with maximum value. Karp [22] showed that MCP is an $\mathcal{NP}$-hard problem. Figure 1a shows an example graph with five vertices and seven edges where the number close to each edge represents the corresponding weight. Figure 1b shows a possible solution, $x = (W, W')$, where $W = \{1, 2\}$ and $W' = \{3, 4, 5\}$. The value of this solution is $cut(W, W') = 9 + 14 + 10 + 5 = 38$, computed as the sum of the edges whose endpoints are in different sets (dashed edges). This solution can be represented as a binary vector $x = \{1, 1, 0, 0, 0\}$ where $x_i = 1$ indicates that the corresponding vertex is in W, while $x_i = 0$ means that the vertex is in W'.

For a given solution x of a MCP, we define $\mathcal{N}_{drop}^{Bin}(x)$ neighborhood using the $drop(x, i)$ move operator. This operator is responsible of changing the value of a variable x_i from 1 to 0, producing a new solution x'. The associated neighborhood, $\mathcal{N}_{drop}^{Bin}(x)$, has size n (in the worst case) and is a subset of $\mathcal{N}_1^{Bin}(x)$. Formally, $\mathcal{N}_{drop}^{Bin}(x)$ is defined as:

$$\mathcal{N}_{drop}^{Bin}(x) = \{x' \leftarrow drop(x, i) : x_i = 1 \wedge 1 \leq i \leq n\}$$

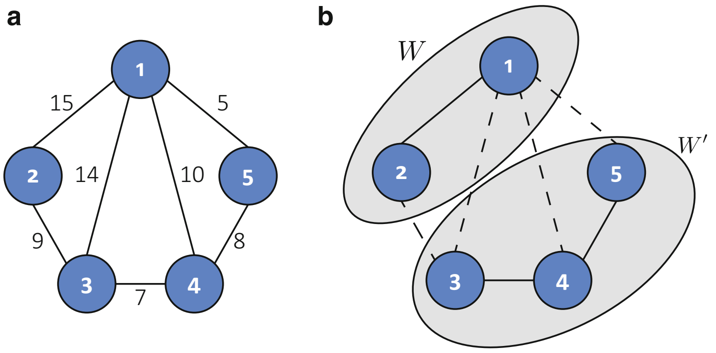

Fig. 1 Example of graph with five vertices and seven edges and a possible solution for the MCP. (**a**) Example of a graph. (**b**) Solution $x = \{1, 1, 0, 0, 0\}$

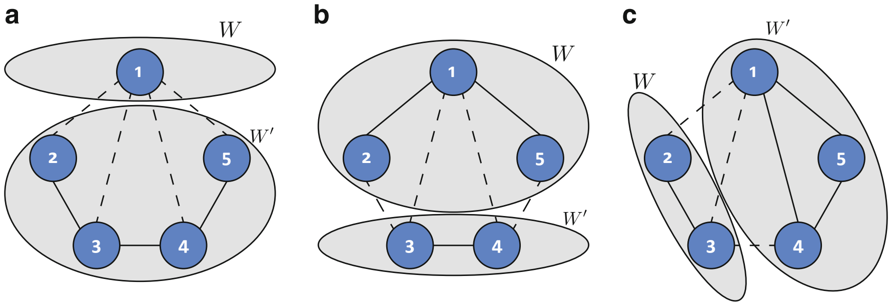

Fig. 2 Solutions generated when performing the different moves to the one depicted in Fig. 1b. (**a**) $x' \leftarrow drop(x, 2)$. (**b**) $x'' \leftarrow add(x, 5)$. (**c**) $x''' \leftarrow swap(x, 1, 3)$

Figure 2a shows an example of a this type of move. In particular, $drop(x, 2)$ considers the solution x (depicted in Fig. 1b), removes the vertex 2 from W, and includes it in W', producing a new solution $x^{drop} = \{1, 0, 0, 0, 0\}$. The Hamming distance between x and x' is 1 since there is only one vertex located in a different group.

Symmetrically, we define the $add(x, i)$ as the move operator responsible of changing the value of a variable x_i from 0 to 1, producing a new solution x^{add}. Considering the MCP, this move operator removes a vertex from W' and includes it in W. This move is represented as $x'' \leftarrow add(x, i)$, and the neighborhood, $\mathcal{N}_{add}^{Bin}(x)$, with size n in the worst case, is:

$$\mathcal{N}_{add}^{Bin}(x) = \{x'' \leftarrow add(x, i) : x_i = 0 \wedge 1 \leq i \leq n\}$$

Figure 2b shows the move $add(x, 1)$. It considers again the solution x (depicted in Fig. 1b), removes the vertex 1 from W' including it in W. The Hamming distance between the new produced solution, $x'' = \{1, 1, 0, 0, 1\}$, and x is again 1.

Finally, we define $\mathcal{N}_{swap}^{Bin}(x)$ neighborhood as a subset of $\mathcal{N}_2^{Bin}(x)$ defined by *swap* move. *Swap* move is defined as an operation that changes the value of one variable x_i from 1 to 0, and simultaneously a different variable x_j change the value from 0 to 1. This move interchanges a vertex from W to W' and, simultaneously a different vertex from W' to W. This move is represented as $x''' \leftarrow swap(x, i, j)$, and the neighborhood, $\mathcal{N}_{swap}^{Bin}(x)$ (with size n^2 in the worst case), is:

$$\mathcal{N}_{swap}^{Bin}(x) = \{x''' \leftarrow swap(x, i, j) : x_i \neq x_j \wedge 1 \leq i, j \leq n\}.$$

Figure 2c shows the move $swap(x, 1, 3)$. It considers again the solution x (depicted in Fig. 1b) and interchanges the vertex 1 and 3 between W and W', respectively. The Hamming distance between the new produced solution, $x''' = \{0, 1, 1, 0, 0\}$, and x is in this case 2.

Neighborhoods for Integer Problems

There are some optimization problems where the solution x is represented as a vector of n variables, $x = (x_1, x_2, \ldots, x_n)$, such that $l \leq x_j \leq u$ for all j (i.e., assuming that all variables are integer and bounded within the interval $[l, u]$). For example, the solutions to the set covering problem [9], the capacitated task allocation problem [24], or the maximally diverse grouping problems [11] can be represented as a vector of integer values. For solving these problems, neighborhood structures $\mathcal{N}_k^{Int}(x)$ are usually induced from the metric defined as:

$$\rho_{int}(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{7}$$

or

$$\rho_{int}(x, y) = \max_{1 \leq i \leq n} |x_i - y_i| \tag{8}$$

The neighborhood $\mathcal{N}_k^{Int}(x)$ denotes the set of solutions in the k–th neighborhood of x, and it is defined as

$$\mathcal{N}_k^{Int}(x) = \{y \in X \mid \rho_{int}(x, y) \leq k\} \tag{9}$$

or

$$\mathcal{N}_k^{Int}(x) = \{y \in X \mid k - 1 \leq \rho_{int}(x, y) \leq k\} \tag{10}$$

We use the maximally diverse grouping problem to illustrate neighborhoods for integer problems. It consists of grouping a set of M elements into G mutually

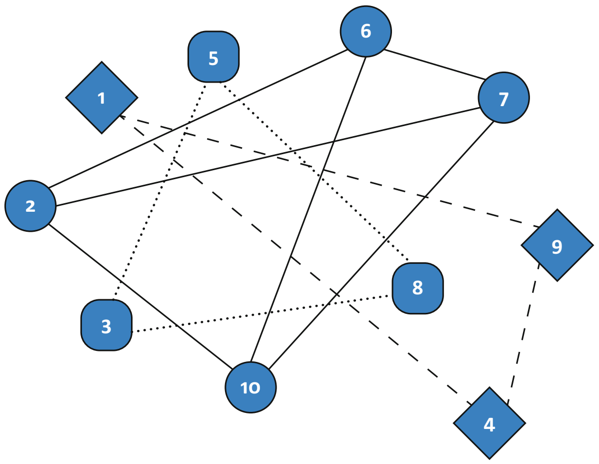

Fig. 3 Example of the maximally diverse grouping problem

disjoint groups in such a way that the diversity among the elements in each group is maximized. The diversity among the elements in a group is calculated as the sum of the individual distances between each pair of elements, where the notion of distance depends on the specific application context. The objective of the problem is to maximize the overall diversity, that is, the sum of the diversity of all groups. Figure 3 shows an example of a graph with ten vertices where we must select three groups of elements. For the sake of clarity, we only represent the distances among selected elements. In addition, we use different shapes to differentiate the vertices on each group. In particular, group 1 (diamond shape) is $G_1 = \{1, 4, 9\}$ whose distances among edges are represented with dashed lines; group 2 (rectangle shape) is $G_2 = \{3, 5, 8\}$ represented with dotted lines; and group 3 (circle shape) is $G_3 = \{2, 6, 7, 10\}$ represented with solid lines. Therefore, this solution can be represented as a vector $x = \{1, 3, 2, 1, 2, 3, 3, 2, 1, 3\}$ (i.e., each vertex i is located in group x_i).

With this representation, we define the following moves:

Exchange of values (Swap move): given a solution x, the exchange of the values of the variables x_i and x_j is denoted by $swap(x, i, j)$ and generates a new solution x' such that $x'_k = x_k$, $\forall k \neq i, j$, and $x'_i = x_j$, $x'_j = x_i$. Note that resulting solution x' belongs to the neighborhood

$$\mathcal{N}_0^{Int}(x) = \{y \in X \mid \max_{1 \leq i \leq n} |x_i - y_i| = 0\}.$$

Replacement of a single value (Replacement move): given a solution x, the replacement move denoted by $replace(x, j, i)$ creates a solution x' for which

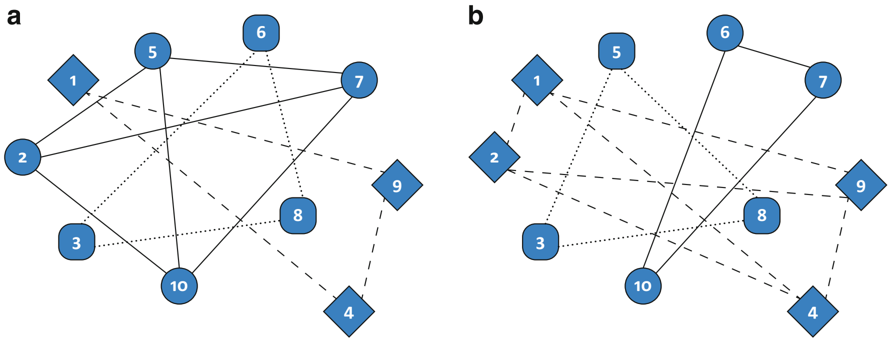

Fig. 4 Example of moves applied to a solution for the maximally diverse grouping problem. (**a**) $x' \leftarrow swap(x, 6, 5)$. (**b**) $x'' \leftarrow replace(x, 2, 1)$

$x'_k = x_k, \forall k \neq j$, and $x'_j = i$. The move is such that $x_j \neq i$. In this case, the resulting solution x' belongs to the neighborhood

$$\mathcal{N}_m^{Int}(x) = \{y \in X \mid \sum_{1 \le i \le n} |x_i - y_i| \le m\},$$

where $m = |x_j - i|$.

Given the solution $x = \{1, 3, 2, 1, 2, 3, 3, 2, 1, 3\}$, the move $swap(x, 6, 5)$ interchanges the corresponding group of vertices 6 and 5, producing the solution $x' = \{1, 3, 2, 1, 3, 2, 3, 2, 1, 3\}$ (see Fig. 4a). Similarly, the move $replace(x, 2, 1)$ includes the vertex 2 in group 1 (removing it from its original group). The final solution in this case is $x'' = \{1, 1, 2, 1, 2, 3, 3, 2, 1, 3\}$ (see Fig. 4b).

Neighborhoods for Permutation Problems

In permutation-based representations, a solution is typically expressed as a labeling (permutation), where each element receives a unique label from 1 to n, being n the size of the problem. The traveling salesman problem [35], the linear ordering problem [30], or the cutwidth minimization problem [34] are examples of this kind of problems. For solving these problems, neighborhood structures $\mathcal{N}_k^{Int}(x)$ may be induced using several metrics (see e.g., [4] for possible metric). However, neighborhoods are usually induced from the Cayley distance:

$$\rho_{perm}(x, y) := \text{the minimum number of transpositions needed to obtain y from x} \tag{11}$$

So, the neighborhood $\mathcal{N}_k^{Perm}(x)$ denotes the set of solutions in the k–th neighborhood of x, and it is defined as:

$$\mathcal{N}_k^{Perm}(x) = \{y \in X \mid \rho_{perm}(x, y) = k\} \tag{12}$$

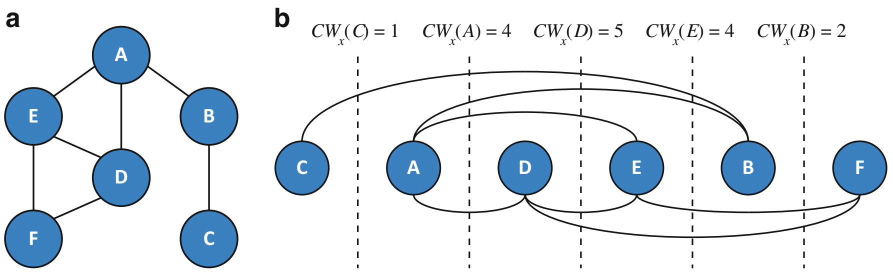

Fig. 5 Graph with six vertices and seven edges and an example solution for the CWP. (**a**) Example graph. (**b**) Cutwidth of G for a labeling x

We use the cutwidth minimization problem (CMP) to illustrate some neighborhoods for permutation-based problem. It can be easily described in mathematical terms. Given a graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, a labeling or linear arrangement x of G assigns the integers $\{1, 2, \ldots, n\}$ to the vertices in V, where each vertex receives a different label. The cutwidth of a vertex v with respect to x, $CW_x(v)$, is the number of edges $(u, w) \in E$ satisfying $x(u) \leq x(v) < x(w)$. In mathematical terms:

$$CW_x(v) = |\{(u, w) \in E : x(u) \leq x(v) < x(w)\}|$$

The cutwidth of G with respect to x is defined as the maximum value of all $CW_x(v)$ for $v \in V$. More formally:

$$CW_x(G) = \max_{v \in V} CW_x(v)$$

The optimum cutwidth of G is then defined as the minimum $CW_x(G)$ over all possible layouts of G. This optimization problem is $\mathcal{NP}$-hard even for graphs with a maximum degree of three [27]. Figure 5a shows an example of an undirected graph with six vertices and seven edges. Figure 5b shows a labeling, x, of the graph in Fig. 5a, setting the vertices in a line with the order of the labeling, as commonly represented in the CMP. We represent x with the ordering (C, A, D, E, B, F) meaning that vertex C is located in the first position (label 1), vertex A is located in the second position (label 2), and so on. In Fig. 5c, the cutwidth of each vertex is represented as a dashed line with its corresponding value. For example, the cutwidth of vertex C is $CW_x(C) = 1$, because the edge (C, B) has an endpoint in C labeled with 1 and the other endpoint in a vertex labeled with a value larger than 1. In a similar way, we can compute the cutwidth of vertex A, $CW_x(A) = 4$, by counting the appropriate number of edges: (C, B), (A, B), (A, E), and (A, D). Then, since the cutwidth of the graph G, $CW_x(G)$, is the maximum of the cutwidth of all vertices in V, in this particular example, we obtain $CW_x(G) = CW_x(D) = 5$.

The associated neighborhoods for this optimization problem are typically based on two different move operators. The first one is referred to as *exchange*. Given a solution $x = (v_1, \ldots, v_i, \ldots, v_j, \ldots, v_n)$, we define $\mathit{exchange}(x, i, j)$ as exchanging in x the vertex in position i (i.e., vertex v_i) with the vertex in position j (i.e., vertex v_j), producing a new solution $x' = (v_1, \ldots, v_{i-1}, v_j, v_{i+1}, \ldots, v_{j-1}, v_i, v_{j+1}, \ldots, v_n)$. So, *exchange* move represents a transposition. For the sake of simplicity, we denote $x' = \mathit{exchange}(x, i, j)$. The associated neighborhood $\mathcal{N}^{Perm}_{exchange}$, obtained applying all possible *exchange* moves, has size $n(n-1)/2$, and it is formally defined as:

$$\mathcal{N}^{Perm}_{exchange}(x) = \{x' \leftarrow \mathit{exchange}(x, i, j) : i \neq j \wedge 1 \leq i, j \leq n\}$$

This neighborhood is actually $\mathcal{N}^{Perm}_1$ neighborhood.

The second move operator for CMP is known as *insert*. Specifically, given a solution x, we define $\mathit{insert}(x, j, v_i)$ as the move consisting of deleting v_i from its current position i and inserting it in position j. This operation results in the new solution x' as follows:

- If $i \geq j$, then $x = (\ldots, v_{j-1}, v_j, v_{j+1}, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots)$, and the vertex v_i is inserted just before the vertex v_j , obtaining $x' = (\ldots, v_{j-1}, v_i, v_j, v_{j+1}, \ldots, v_{i-1}, v_{i+1}, \ldots)$.
- If $i < j$, then $x = (\ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_{j-1}, v_j, v_{j+1}, \ldots)$, and the vertex v_i is inserted just after the vertex v_j , obtaining $x' = (\ldots, v_{i-1}, v_{i+1}, \ldots, v_{j-1}, v_j, v_i, v_{j+1}, \ldots)$.

Thus, a move $\mathit{insert}(x, j, v_i)$ generates a solution belonging to the neighborhood $\mathcal{N}^{Perm}_{|j-i|}(x)$. The associated neighborhood, $\mathcal{N}^{Perm}_{insert}$, obtained applying all possible *insert* moves has size $n(n-1)/2$, and it is formally defined as:

$$\mathcal{N}^{Perm}_{insert}(x) = \{x' \leftarrow \mathit{insert}(x, j, v_i) : i \neq j \wedge 1 \leq i, j \leq n\}$$

The following example illustrates how the *insert* is implemented. Let $x = (C, A, D, E, B, F)$ be a solution of the cutwidth problem. Suppose that we perform $\mathit{insert}(x, 2, B)$, obtaining solution $x' = (C, B, A, D, E, F)$. Figure 6 graphically shows the solutions before and after the move.

Local Search Methods

Local search methods are likely the oldest and simplest heuristic methods. Starting from a given feasible solution, these procedures explore a determined neighborhood in each iteration, replacing the current solution if a neighbor improves the objective function. The search ends when all neighbor solutions are worse (i.e., larger objective function value in a minimization problem) meaning that a local optimum is found.

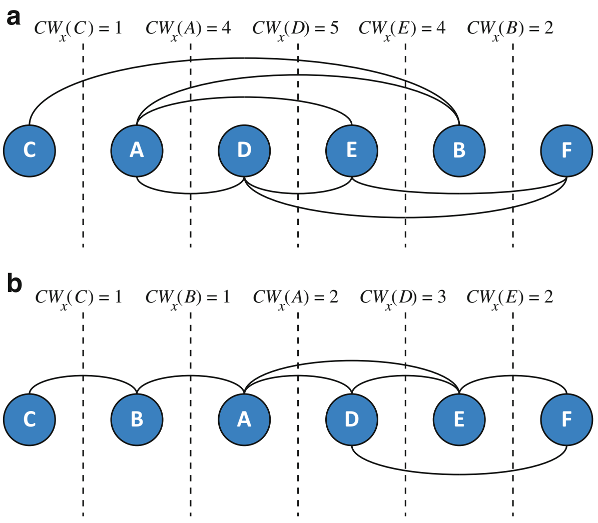

Fig. 6 Example of performing an insert move in a permutation solution. (**a**) Solution x before the move. (**b**) Resulting solution after $insert(x, 2, B)$

Algorithm 1: Best improvement(x)

```
1:  $improve \leftarrow \texttt{true}$
2:  while ($improve$) do
3:      $improve \leftarrow \texttt{false}$
4:      $x' \leftarrow \arg\min_{x \in N(x)} f(x)$
5:      if $f(x') < f(x)$ then
6:          $x \leftarrow x'$
7:          $improve \leftarrow \texttt{true}$
8:      end if
9:  end while
10: return $x$
```

There exist two typical strategies to explore the corresponding neighborhood: **best improvement** and **first improvement**. In the former (also known as **steepest descent**), the associated neighborhood is completely explored by a fully deterministic procedure, performing the best associated move. Therefore, no matter how the neighborhood is scanned, since all neighbor solutions are visited. Algorithm 1 shows the typical pseudocode of this local search method for a minimization problem. The algorithm starts by initializing the control variable *improve* (step 1). Then, the best improvement strategy performs iterations until it finds a local optimum with respect to neighborhood $N(x)$ (steps 2–9). Given a solution x, the best neighbor solution x' is determined in step 4. This instruction has a computational complexity of $|N(x)|$. In steps 5–8, it is decided whether to perform a new iteration (by updating x to x') or not (abandoning the search).

The first improvement strategy tries to avoid the time complexity of exploring the whole neighborhood by performing the first improving move encountered during the exploration of the corresponding neighborhood. In this kind of exploration, the order in which the neighbors are inspected can have a significant influence of the efficiency of the search. Instead of using a fixed ordering for scanning the neighbors, random orders are usually suggested, since the first scanning strategy always drives to the same local optimum, while the second one can reach different local optima.

The pseudocode of this strategy is equivalent to the one presented in Algorithm 1. The only difference is in step 4, where, instead of selecting the best neighbor, it selected the first neighbor which improves the incumbent solution (in terms of objective function value). Additionally, if we follow a predefined order (e.g., a lexicographic order), the first positions in that order (e.g., $1, 2, \ldots,$ in the lexicographic order) are favored, producing a kind of bias in the search.

Figure 7 shows the performance of two iterations of both strategies starting from the same solution, where the numbers over the arrows indicate the order of exploration of the solutions. Specifically, considering the MCP described in section "Neighborhoods for Binary Problems", the initial solution is $x_1 = [1, 1, 0, 0, 0]$, representing $W = [1, 2]$ and $W' = [3, 4, 5]$, with an objective function value of 38. The neighborhood selected to be explored is $\mathcal{N}_{add}^{Bin}(x_1)$, where each neighbor is generated by adding a new vertex to W (removing it from W'). In the first iteration, the best improvement strategy (Fig. 7a) generates all possible neighbor solutions for x_1 (i.e., x_2, x_3, x_4). The exploring order, as stated before, is irrelevant, since it is going to explore the whole neighborhood. Then, the method selects the best neighbor solution, which is x_3 in Fig. 7a, with a higher (better) objective function value of 43. Finally, in the next iteration, the method explores the entire x_3 neighborhood, stopping after the exploration, since there is no better neighbor solution.

Regarding the first improvement method (Fig. 7b), the exploration order (which is relevant in this case) is selected at random, starting with solution x_2. As the objective function value of x_2 is lower (worse) than x_1, the method explores another neighbor, which is x_4. The objective function value for x_4, 41, is better than for x_1, 38, so in this case, the first improvement strategy stops the iteration, starting the next one from this new best solution x_4. Notice that although solution x_3 is better than x_4, it is not explored in this strategy, since the order selected has lead the method to find a better solution before reaching x_3. Finally, in the next iteration, the method starts from x_4 and then explores, in this order, x_6 and x_5, stopping the search since there is no improvement in any neighbor.

In the context of large neighborhoods, there is a compromise between the number of iterations required to find a local optimum and the associated computing time. In general, iterations performed in the first improvement strategy are more efficient than those in the best improvement one, since the former only evaluates part of the neighborhood, while the latter explores it completely. On the other hand, the improvement obtained in the first improvement strategy is typically smaller than the one achieved by the best improvement strategy, requiring in general more iterations to obtain the local optimum. Additionally, the best improvement strategy

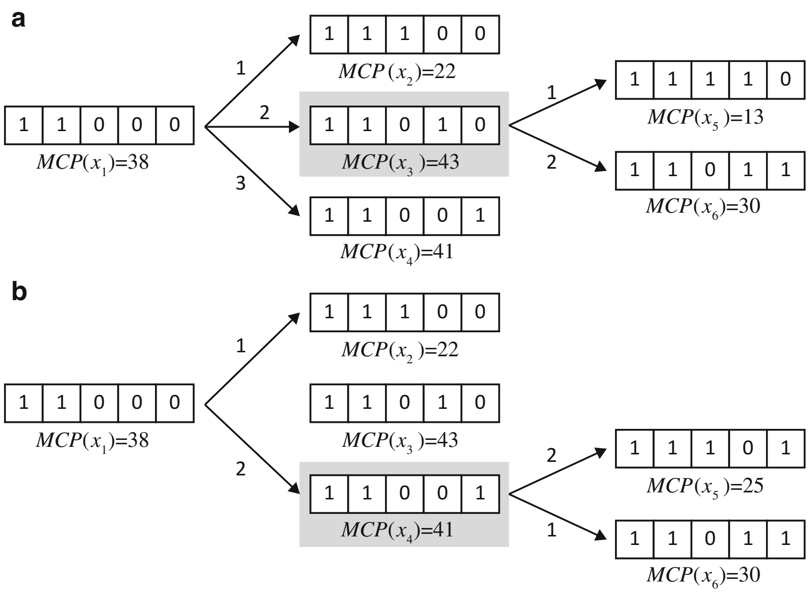

Fig. 7 Comparison of best and first improvement strategies when starting from the same solution for the MCP. (**a**) Best improvement. (**b**) First improvement

is usually more adequate to perform efficient catching and updating mechanisms, which allows the search to efficiently explores the neighborhood [20].

In [17], an empirical study on traveling salesman problem was conducted in order to compare the first and the best improvement strategy within 2-opt neighborhood structure. It appeared that the quality of the final solution depends on the quality of the initial solution: (i) if random initial solution is chosen, the better and faster is the first improvement strategy; (ii) the opposite holds if the greedy solution is taken as initial one.

VND Variants

The Variable Neighborhood Search (VNS) is a metaheuristic proposed in [32] as a general framework to solve hard problems. It is based on a simple idea: systematical changes of neighborhood structures within the search procedure. Let $\mathcal{N} = \{\mathcal{N}_1, \ldots, \mathcal{N}_{k_{\max}}\}$ be set of operators such that each operator $\mathcal{N}_k$, $1 \leq k \leq k_{\max}$ maps a given solution x to a neighborhood structure $\mathcal{N}_k(x)$. Note that the order of operators taken from the set $\mathcal{N}$ defines also the order of neighborhood structures of a given solution x examined. When solving an optimization problem by using different neighborhood structures, VNS methodology proposes to explore them in three different ways: (i) at random, (ii) deterministically, or (iii) mixed (both, in deterministic and random fashion).

Variable neighborhood descent (VND) is a variant of VNS that explores neighborhoods in a deterministic way. In general, VND explores small neighborhoods until a local optimum is encountered. At that point, the search process switches to a different (typically larger) neighborhood that might allow further progress. This approach is based on the fact that a local optimum is defined with respect to a neighborhood relation, such that if a candidate solution x is locally optimal in a neighborhood $\mathcal{N}_i(x)$, it is not necessarily a local optimum for another neighborhood $\mathcal{N}_j(x)$. Thus, VND explores the solution space using several neighborhood structures either in a (i) sequential, (ii) a nested (or composite), or (iii) mixed nested way [21, 43].

Sequential Variable Neighborhood Descent Procedures

Most typical VND variants traverse the list of neighborhood structures in a sequential way. Within this variants, the basic VNS, the pipe VND, the cyclic VND, and the union VND emerge as the most representative search procedures. These variants differ in how they implement the neighborhood change procedures. Specifically, if an improvement has been detected in some neighborhood, this is how the search (after updating the incumbent solution) is continued:

- **Basic VND (B-VND)** – returns to the first neighborhood from the list
- **Pipe VND (P-VND)** – continues the search in the same neighborhood
- **Cyclic VND (C-VND)** – resumes the search in the next neighborhood from the list
- **Union VND (U-VND)** (sometimes called multiple neighborhood search [44], where the single neighborhood is obtained as the union of all predefined neighborhoods) – continues the search in the same large neighborhood. U-VND is recently proposed in [2, 26, 44] and used within Tabu search.

All these VND procedures follow the steps given in Algorithm 2 and start from a given solution x. In each VND iteration, a local search procedure through a given neighborhood structure is applied, followed by a neighborhood change function (Step 7). The neighborhood change function defines the neighborhood structure that will be examined in the next iteration. Each VND variant stops when there is no improvement with respect to any of the considered neighborhood structures. Thus, the solution obtained by any sequential VND is a local optimum with respect to all neighborhood structures.

We show the performance of the three aforementioned variants considering the example introduced in [31]. In particular, it shows an empirical study on different VND variants used to solve traveling salesman problem. For testing purposes, 15,200 random test instances were generated in the way described in [17]. Within VND variants, three classical TSP neighborhood structures are considered: 2-opt (Fig. 8), Insertion-1 (Fig. 9), and Insertion-2 (Fig. 10). On each instance from this data set, each VND variant, except U-VND, is tested under 24 different settings.

Algorithm 2: Sequential variable neighborhood descent

```
   Function SeqVND ($x$,$k_{\max}$,$\mathcal{N}$)
1      $x'' \leftarrow x$
2      Stop= False
   while Stop= False do
3      $x \leftarrow x''$
4      Stop= True
5      $k \leftarrow 1$
       while $k \leq k_{\max}$ do
6          $x' \leftarrow \arg\min_{y \in \mathcal{N}_k(x)} f(y)$
7          Change Neighborhood ($x, x', k$)
8          if $f(x') < f(x'')$ then
9              $x'' \leftarrow x'$
10             Stop= False
           end
       end
   end
11 return $x''$;
```

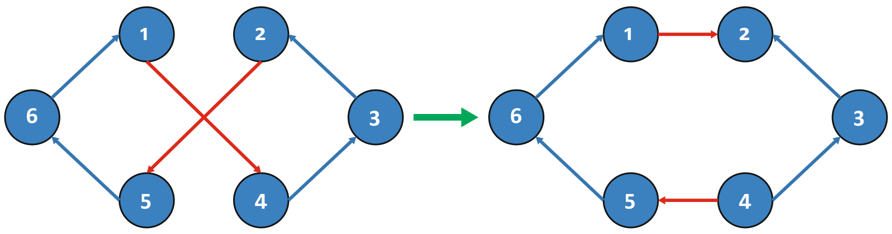

Fig. 8 Example of a 2-opt move involving vertices 1, 2, 4, and 5

Each setting corresponds to choosing the following: (i) one out of two common ways for getting an initial solution, at *random* (solution generated as a random permutation of nodes) or *greedy*; (ii) one out of six possible neighborhood orders; and (iii) the *best* or the *first improvement search strategy*. This gives $2 \times 6 \times 2 = 24$ different search methods that use 2-opt, Insertion-1, and Insertion-2 neighborhoods. On the other hand, U-VND is tested under only two different settings as: U-VND that uses the best improvement search strategy and the greedy initial solution and U-VND that uses the best improvement search strategy and the random initial solution. Note that if the first improvement search strategy is used within U-VND, then U-VND is equivalent to basic VND.

Table 1 summarizes results considering the entire set of 15,200 instances as a test case. Namely, the average solution values (Column '`av. best value`') and

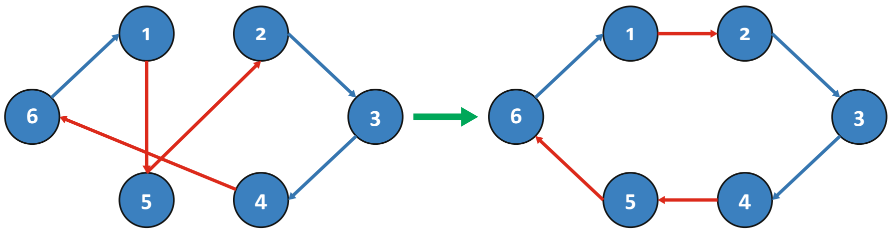

Fig. 9 Example of an Insertion-1 move where vertex 5 is inserted between vertices 1 and 2 and removed from its corresponding former position

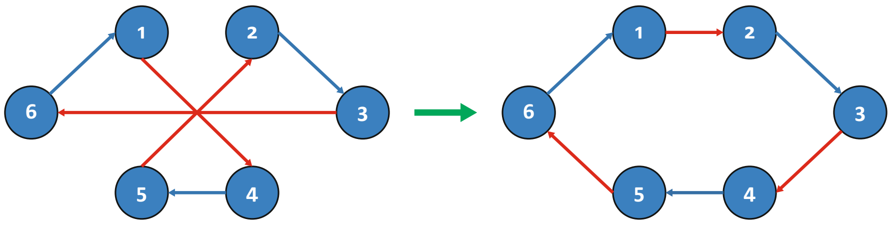

Fig. 10 Example of an Insertion-2 move where vertices 4 and 5 are inserted between vertices 1 and 2, removing them from its corresponding former position

Table 1 Comparison of VND variants

VND variant	av. best value	av. time (s)
Basic VND	1198.24	0.16
Pipe VND	1198.52	0.12
Cyclic VND	1198.76	0.46
Union VND	1197.65	1.06

average CPU times (Column '`av. time`') over all test instances in the data set, attained by VND variants under the best settings, are reported.

From the results presented in Table 1, the following conclusions may be drawn: (i) U-VND is slightly better than the others VNDs, regarding the best average values attained, but much slower than the others. This is explained by the fact that U-VND in each iteration performs exploration of large part of the solution space before deciding to re-center the search. Obviously, such principle is suitable for reaching good final solution, but requires a large CPU time; (ii) comparing VNDs that re-center the search in the inner loop (i.e., B-VND, P-VND, and C-VND), it follows that B-VND is able to provide the best solution values. Regarding average CPU times consumed by B-VND, P-VND, and C-VND to find the best-reported average solution value, their ranking is as follows: the fastest is P-VND, B-VND is ranked as the second, while C-VND is the slowest one. However, since, B-VND consumed negligible more CPU time to find the best reported average solution value comparing to CPU time that P-VND consumed to do so, we may conclude that B-VND is the most appropriate sequential VND version.

Nested Variable Neighborhood Descent

A nested (composite) variable neighborhood descent procedure [21] explores a large neighborhood structure obtained as a composition of several neighborhoods. More precisely, let $\mathcal{N} = \{N_1, \ldots N_{k_{\max}}\}$ again be set of move operators such that each one N_k, $1 \leq k \leq k_{\max}$ maps a given solution x to a predefined neighborhood structure $N_k(x)$. Then, the neighborhood explored within a nested variable neighborhood procedure is defined with operator $N_{nested} = N_1 \circ N_2 \circ \cdots \circ N_{k_{\max}}$. More precisely, the composite neighborhood of solution x is formed by first applying the move operator N_1, obtaining $N_1(x)$. Then, the move operator N_2 is applied to all solutions in $N_1(x)$, forming the set $N_1(N_2(x))$ and so on. Obviously, the cardinality of a neighborhood structure $N_{nested}(x) = N_1(N_2(\ldots(N_{k_{\max}}(x))))$ of some solution x is less than or equal to the product of cardinalities of nested (composed) neighborhoods, i.e.,

$$|N(x)| \leq \prod_{k=1}^{k_{\max}} |N_k(x)|.$$

Such cardinality obviously increases chances to find an improvement in the neighborhood. The nested VND is illustrated in Algorithm 3. The neighborhood $N(x)$ may be explored by using either the first or the best improvement search strategy. However, since its cardinality is usually very large, the first improvement is used more often [21, 43].

Algorithm 3: Steps of best improvement nested VND

```
Function Nested_VND($x$, $k_{\max}$, $\mathcal{N}$)
$N_{nested} = N_1 \circ N_2 \circ \cdots \circ N_{k_{\max}}$
repeat
    $x' \leftarrow x$;
    $x \leftarrow \text{argmin}_{y \in N(x')} f(y)$;
until $f(x') \leq f(x)$;
return $x'$;
```

We use the uncapacitated r-allocation p-hub median problem (r-p-HMP) to illustrate how this VND strategy works. Specifically, the r-p-HMP may be stated as follows. Given n nodes, this problem considers for each pair of nodes i and j, the distance d_{ij} and the amount of flow $t_{ij} \geq 0$ that needs to be transferred from i to j. It is generally assumed that the transportation between non-hub nodes i and j is only possible via hub nodes h_i and h_j, to which nodes i and j are assigned, respectively. Transferring t_{ij} units of flow through path $i \to h_i \to h_j \to j$ induces a cost $c_{ij}(h_i, h_j)$, which is computed as

$$c_{ij}(h_i, h_j) = t_{ij}(\gamma d_{ih_i} + \alpha d_{h_i h_j} + \delta d_{h_j j}).$$

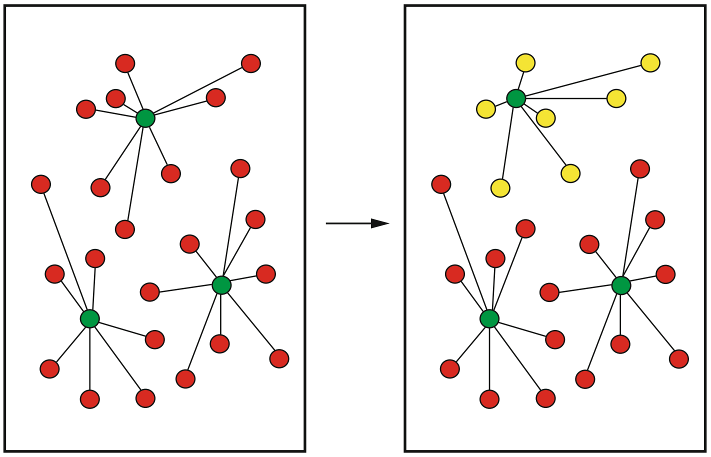

Fig. 11 Interchange neighborhood $\mathcal{N}_H$. *Yellow dots* are the possible new hub location for the one represented in *green*

Parameters γ, α, and δ represent unit rates for collection (origin-hub), transfer (hub-hub), and distribution (hub-destination), respectively. Note that the hub nodes h_i and h_j may be equal.

We represent a solution of r-p-HMP by a set H containing p hubs and a matrix A, where each row i contains r hubs assigned to node i (i.e., i-th row coincides with the set H_i). Thus, our solution is represented as $x = (H, A)$. The initial solution is obtained using the greedy heuristic described in [38]. We consider two neighborhood structures of a given solution $x = (H, A)$. The first neighborhood structure, denoted by $\mathcal{N}_H$, is obtained by replacing one hub node from H by another non-hub node from $N \setminus H$ (see Fig. 11). More formally,

$$\mathcal{N}_H(x) = \{x' \mid x' = (H', A'), |H \cap H'| = p - 1\}.$$

The second neighborhood, denoted by $\mathcal{N}_A$, is obtained by replacing one hub assigned to some node with another hub, while the set H remains unchanged (see Fig. 12):

$$\mathcal{N}_A(x) = \{x'|x' = (H, A'), |A \backslash A'| = 1\}.$$

Unfortunately, evaluating the objective function value of a solution from $\mathcal{N}_H$ requires to solve an allocation problem, which is $\mathcal{NP}$ hard by itself [25]. Therefore, solving exactly the associated allocation problem will be quite time consuming. In order to deal with this drawback, we find near-optimal allocation

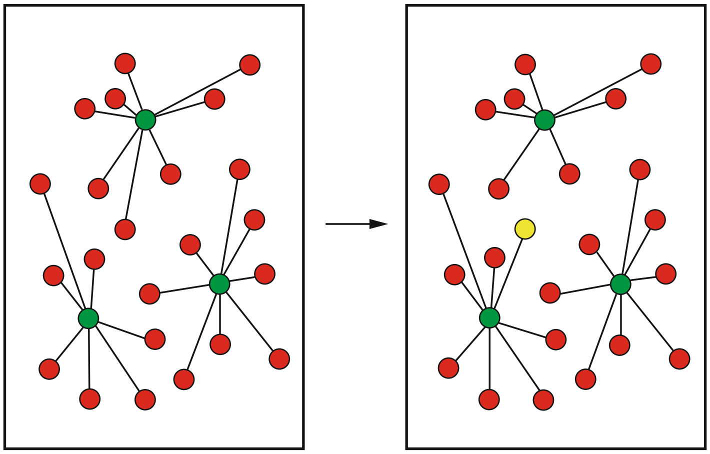

Fig. 12 Allocate neighborhood $\mathcal{N}_A$. *Yellow dot* represents the non-hub node which is reallocated to the another hub

A' of some solution $x'' \in \mathcal{N}_H$ as follows. For each node, we firstly determine node-to-hub allocation using the greedy procedure (see Algorithm 4).

Algorithm 4: Greedy allocation

```
   Function GreedyAllocation(H, A)

1  for $i \in N$ do
2      for $j = 1$ to $p$ do $value(j) = d_{ih_j} \sum_{k \in N} t_{ik} + \sum_{k \in N} t_{ik} d_{h_j k}$;

3      Sort array $value$ in nondecreasing order i.e.,
       $value(\pi(1)) \le value(\pi(2)) \le \cdots \le value(\pi(p))$;

4      for $j = 1$ to $r$ do $A[i][j] = h_{\pi(j)}$;
   end
```

The solution $x'' = (H'', A'')$ obtained in this way is then improved by exploring the second neighborhood $\mathcal{N}_A(x'')$. In that way, the so-called nested variable neighborhood descent (Nest-VND) is defined (see Algorithm 5).

Algorithm 5: Nested VND for r-p-HMP

```
   Function NestVND(x);

1  for each $x'' \in N_H(x)$ do
2      GreedyAllocation(H″, A″);
3      Select $x'$ as the best solution in $N_A(x'')$;
4      if $x'$ is better than $x$ then $S \leftarrow x'$
   end
5  return x
```

Table 2 Comparison of GRASP and GVNS on AP instances

	GRASP		GVNS_RP		
n	Aver. value	Aver. time	Aver. value	Aver. time	% impr.
60	122348.90	4.59	121829.27	3.73	0.42
65	123001.53	6.66	122689.74	5.87	0.25
70	123931.76	10.51	123604.38	5.75	0.26
75	124776.42	11.11	124650.73	5.93	0.10
80	125148.22	14.40	124844.76	9.36	0.24
85	125566.58	19.48	125378.23	13.10	0.15
90	124934.99	22.95	124734.55	12.32	0.16
95	125121.18	24.27	124926.55	25.45	0.16
100	125805.04	4.81	125588.19	10.39	0.17
150	126728.85	21.42	126307.10	24.70	0.33
200	129144.44	58.86	128788.66	98.67	0.28
Avg.	125137.08	18.10	124849.29	19.57	0.23

In [43], the above nested VND was used as a local search within a general variable neighborhood search (GVNS)-based heuristic. The performance of this heuristic, named `GVNS_RP`, was compared with the GRASP heuristic proposed in [38]. In Table 2, we provide a summarized results of comparison of `GVNS_RP` and GRASP on instances from AP data set (see [43]). The average value of best found solutions and average CPU times needed for finding these solutions over all instances with the same number of nodes are reported. The column headings are defined as follows. In the first column of Table 2, we report the number of nodes in the considered instances, whereas in the columns "GRASP" and "`GVNS_RP`," we provide the average of best solution values found by GRASP and `GVNS_RP`, respectively. In columns "`time`," the average time needed to reach best found solutions for instances with n nodes is given, while in column "`impr. (%)`," we report the percentage improvement obtained by `GVNS_RP` compared with the current best known values. From the reported results, it follows that within each set of instances with the same number of nodes, there is at least one instance where the best known solution is improved by `GVNS_RP`. Moreover, the average improvement on AP data set achieved by GVNS variants is around 0.25 %.

Mixed Variable Neighborhood Descent

Mixed variable neighborhood descent [21] combines ideas of sequential and nested variable neighborhood descent. Namely, it uses a set of move operators $\mathcal{N} = \{N_1, \dots N_b\}$ to define a nested neighborhood, and, after that, on each element in this nested neighborhood, it applies a sequential variable neighborhood descent variant defined by a set of move operators $\mathcal{N}' = \{N_{b+1}, \dots N_{k_{\max}}\}$. The cardinality of the set explored in one iteration of a mixed VND is bounded by:

$$|N_{mixed}(x)| \leq \prod_{\ell=1}^{b} |N_\ell(x)| \times \sum_{\ell=b+1}^{k_{\max}} |N_\ell(x)|, x \in S.$$

In Algorithm 6, we show the pseudocode of a mixed VND. Note that if the set $\mathscr{N}$ is the empty set (i.e., $b = 0$), we get pure sequential VND. If $b = k_{\max}$, we get pure nested VND. Since nested VND intensifies the search in a deterministic way, boost parameter b may be seen as a way of balancing intensification and diversification in deterministic local search with several neighborhoods.

Algorithm 6: Steps of mixed VND

```
Function Mixed_VND($x$, $b$,$k_{\max}$, $\mathscr{N}$, $\mathscr{N}'$ )
$N = N_1 \circ N_2 \circ \cdots \circ N_b$  $x' \leftarrow x$;
repeat
    $stop = true$;
    $x \leftarrow x'$;
    for each $y \in N(x)$ do
        $x'' \leftarrow$ SeqVND($y$, $k_{\max} - b$, $\mathscr{N}'$) ;
        if $f(x'') < f(x')$ then
            $stop = false$  $x' \leftarrow x''$;
        end
    end
until $stop = true$;
return $x'$;
```

In [21], two mixed VND heuristics along with one basic sequential VND heuristic were proposed for solving the incapacitated single allocation p-hub median problem (USApHMP). Neighborhood structures examined within these VND variants are cluster based. A cluster represents one hub with all locations assigned to it. In particular, the following neighborhood structures are distinguished: (i) *Allocate*- change membership of a non-hub node by reallocating it to the another hub without changing the location of hubs; (ii) *Alternate*-change the location of the hub in one cluster; (iii) *Locate* - select one cluster C with hub h and a location node that is not in this cluster. The selected node becomes a hub, and all locations from the cluster C are assigned to the closest hub (including the new one).

The proposed basic sequential VND heuristic, named `Seq-VND`, examines neighborhood structures Allocate, Alternate, and Locate in that order. On the other hand, the first mixed VND heuristic, named `Mix-VND1`, takes several random points from *Locate* neighborhood and starting from each of them carries out search applying a basic sequential VND heuristic `Seq-VND1`. The used `Seq-VND1` explores *Allocate* and *Alternate* neighborhoods in that order. The second mixed VND heuristic, named `Mix-VND2`, performs more thoroughly exploration of the solution space than `Mix-VND1` applying `Seq-VND1` on each point from *Locate* neighborhood. These three VND variants together with three local searches in the three defined neighborhood structures (Allocate, Alternate, and Locate) are

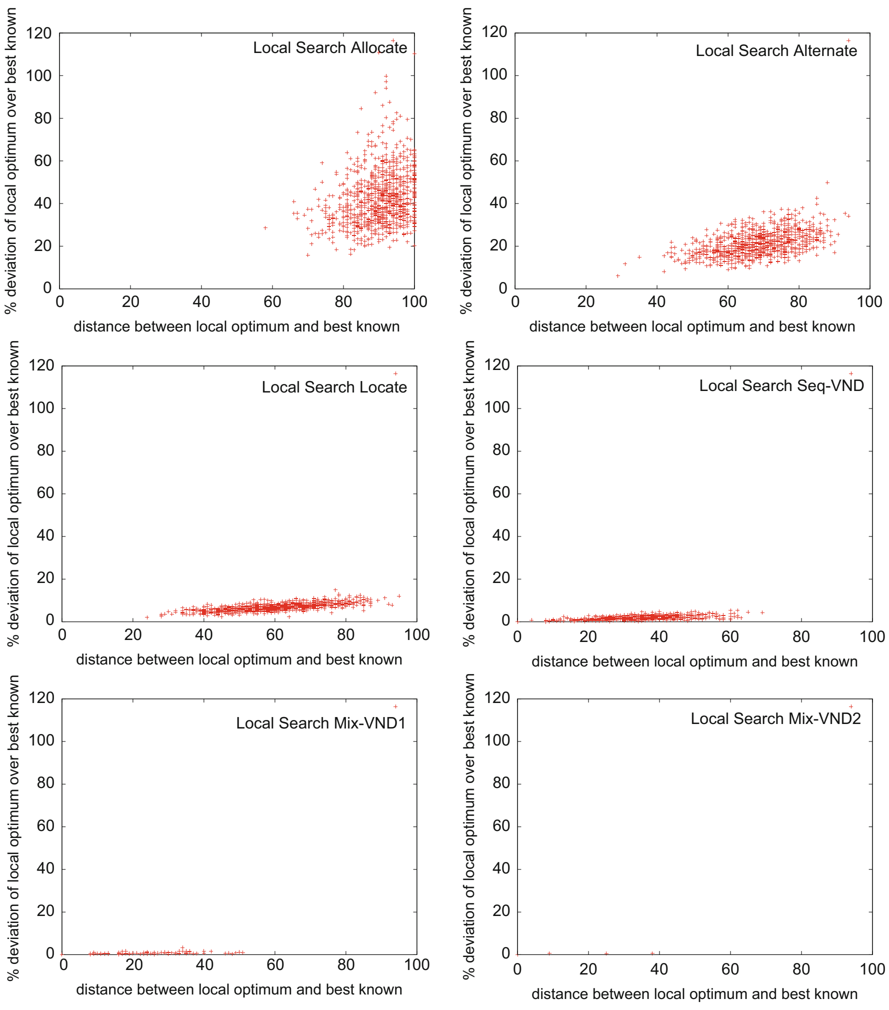

Fig. 13 Multistart on 1000 random initial solutions for AP instance with $n = 100$ and $p = 15$

experimentally compared on AP instance with $n = 100$ and $p = 15$ (see [21]). One thousand initial points are generated at random in the solution space, and a local search is conducted from each one in each type of neighborhood. The distance-to-target diagram [16] presented at Fig. 13 shows the distribution of local minima. Each point (x, y) plots the distance x and percentage deviation y of the local minimum from the best known solution. The results are summarized in Table 3.

Comparing the results in Table 3, we observe that (i) the search in first (*Allocate*) and second (*Alternate*) local neighborhoods is faster than the others, but they have the worse quality; (ii) `Seq-VND` gives better results than the third local search (*Locate*) and is considerably faster; (iii) all three multistart VND local searches

Table 3 Comparison of six local searches on AP instance with $n = 100$ and $p = 15$

	`Allocate`	`Alternate`	`Locate`	`Seq-VND`	`Mix-VND1`	`Mix-VND2`
Average % dev	41.28	21.17	7.41	1.62	0.35	0.16
Minimum % dev.	15.87	6.14	2.58	0.00	0.00	0.00
Maximum % dev.	110.63	49.84	14.6	5.37	3.41	0.68
Average CPU time (sec)	0.002	0.004	0.15	0.06	3.36	27.15

found the best known solution in the AP instance; and (iv) the mixed VND versions give the best results, although, as expected, solution times are longer than solution time of `Seq-VND`.

We see in Fig. 13 that the number of local minima is reduced when VND is used. This is due to the larger neighborhood in VND. For example, the `Seq-VND` neighborhood clearly contains each of the individual neighborhoods, *Allocate*, *Alternate*, and *Locate*. It is remarkable that `Mix-VND2`, which utilizes the largest neighborhood, yields only four local minima for this problem instance.

Conclusions

Local search represents one of the most popular classical heuristic technique that improves (locally) the current feasible solution of some continuous or discrete optimization problem. For that purposes, usually only one neighborhood structure is defined and explored to improve the incumbent solution. Searching for the better solution in such neighborhood is repeated until there is no better solution, i.e., until the local minimum with respect to that predefined neighborhood is reached.

Since the local minimum with respect to one neighborhood structure is not necessary local in another, one needs to construct deterministic local search in cases when more than one neighborhood is used. Such procedures are known as variable neighborhood descent (VND). In this VND survey, we first propose a possible classification of neighborhood structures in solving optimization problems and also provide some simple examples to clearly illustrate the basic ideas. Then, we discuss possible general ways of combining several neighborhoods in the deterministic fashion. Needless to say that the number of possible combination of VND local search variants is large, and they could include problem specific knowledge in building heuristic for each particular problem. Therefore, VND area is an open avenue for the future research in the area of optimization.

Cross-References

▶ Guided Local Search
▶ Multi-Start methods

▶ Restart Strategies
▶ Theory of Local Search

Acknowledgments The works of Nenad Mladenović and Raca Todosijević are partly supported by the Ministry of Education and Science, Republic of Kazakhstan (Institute of Information and Computer Technologies), project number 0115PK00546, and also by the Ministry of Education, Science and Technological Development of Serbia, project number 174010. The works of Abraham Duarte and Jesús Sánchez-Oro are partly supported by the Spanish "Ministerio de Economía y Competitividad" and by "Comunidad de Madrid" with grants refs. TIN2012-35632-C02 and S2013/ICE-2894, respectively.

References

1. Brimberg J, Hansen P, Mladenović N (2015) Continuous optimization by variable neighborhood search. In: Wiley encyclopedia of operations research and management science. Wiley, Hoboken, p 1–13. https://doi.org/10.1002/9780470400531.eorms1107
2. Carrasco R, Pham A, Gallego M, Gortázar F, Martí R, Duarte A (2015) Tabu search for the maxmean dispersion problem. Knowl-Based Syst 85:256–264
3. Cook WJ, Cunningham WH, Pulleyblank WR, Schrijver A (1997) Combinatorial optimization. Wiley, Chichester
4. Deza M, Huang T (1998) Metrics on permutations, a survey. J Comb Inf Syst Sci 23:173–185
5. Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evolut Comput 1(1):53–66
6. Duarte A, Escudero LF, Martí R, Mladenović N, Pantrigo JJ, Sánchez Oro J (2012) Variable neighborhood search for the vertex separation problem. Comput Oper Res 39(12):3247–3255
7. Duarte A, Martí R (2007) Tabu search and GRASP for the maximum diversity problem. Eur J Oper Res 178(1):71–84
8. Duarte A, Sánchez A, Fernández F, Cabido R (2005) A low-level hybridization between memetic algorithm and VNS for the max-cut problem. In: ACM genetic and evolutionary computation conference, New York
9. Feige U (1998) A threshold of Ln N for approximating set cover. J ACM 45(4):634–652
10. Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. J Glob Optim 6:109–133
11. Gallego M, Laguna M, Martí R, Duarte A (2013) Tabu search with strategic oscillation for the maximally diverse grouping problem. J Oper Res Soc 64(5):724–734
12. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman & Co., New York
13. Glover F (1986) Future paths for integer programming and links to artificial intelligence. Comput Oper Res 13(5):533–549
14. Glover F (1998) A template for scatter search and path relinking. In: Selected papers from the third European conference on artificial evolution, AE'97. Springer, London, pp 3–54
15. Hansen P, Mladenović N (1999) An introduction to variable neighborhood search. In: Meta-Heuristics. Springer, Boston, pp 433–458
16. Hansen P, Mladenović N (2003) Variable neighborhood search. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics. Kluwer Academic Publisher, New York, pp 145–184
17. Hansen P, Mladenović N (2006) First vs. best improvement: an empirical study. Discret Appl Math 154(5):802–817
18. Hansen P, Mladenović N, Todosijević R, Hanafi S (2016) Variable neighborhood search: basics and variants. EURO J Comput Optim 1–32. https://doi.org/10.1007/s13675-016-0075-x
19. Holland JH (1992) Adaptation in natural and artificial systems. MIT Press, Cambridge, MA

20. Hoos H, Süttzle T (2004) Stochastic local search: foundations & applications. Morgan Kaufmann Publishers Inc., San Francisco
21. Ilić A, Urošević D, Brimberg J, Mladenović N (2010) A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. Eur J Oper Res 206(2):289–300
22. Karp RM (1972) Reducibility among combinatorial problems. In: Complexity of computer computations. The IBM research symposia series. Springer, New York, pp 85–103
23. Laarhoven PJM, Aarts EHL (1987) Simulated annealing: theory and applications. Kluwer Academic Publishers, Norwell
24. Laguna M, Gortázar F, Gallego M, Duarte A, Martí R (2014) A black-box scatter search for optimization problems with integer variables. J Glob Optim 58(3):497–516
25. Love RF, Morris JG, Wesolowski GO (1988) Facilities location: models and methods. Elsevier Science Publishing Co., New York
26. Lü Z, Hao JK, Glover F (2011) Neighborhood analysis: a case study on curriculum-based course timetabling. J Heuristics 17(2):97–118
27. Makedon FS, Papadimitriou CH, Sudborough IH (1985) Topological bandwidth. SIAM J Algebr Discret Methods 6(3):418–444
28. Martello S, Toth P (1990) Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Inc., New York
29. Martí R, Duarte A, Laguna M (2009) Advanced scatter search for the max-cut problem. INFORMS J Comput 21(1):26–38
30. Martí R, Reinelt G, Duarte A (2012) A benchmark library and a comparison of heuristic methods for the linear ordering problem. Comput Optim Appl 51(3):1297–1317
31. Mjirda A, Todosijević R, Hanafi S, Hansen P, Mladenović N (2016) Sequential variable neighborhood descent variants: an empirical study on travelling salesman problem. Int Trans Oper Res. https://doi.org/10.1111/itor.12282
32. Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24(11):1097–1100
33. Moscato P (1993) An introduction to population approaches for optimization and hierarchical objective functions: a discussion on the role of tabu search. Ann Oper Res 41(1–4):85–121
34. Pantrigo JJ, Martí R, Duarte A, Pardo EG (2012) Scatter search for the cutwidth minimization problem. Ann Oper Res 199(1):285–304
35. Papadimitriou CH (1977) The Euclidean travelling salesman problem is NP-complete. Theor Comput Sci 4(3):237–244
36. Papadimitriou CH, Steiglitz K (1998) Combinatorial optimization: algorithms and complexity. Dover, Mineola
37. Pardo EG, Mladenović N, Pantrigo JJ, Duarte A (2013) Variable formulation search for the cutwidth minimization problem. Appl Soft Comput 13(5):2242–2252
38. Peiró J, Corberán A, Martí R (2014) GRASP for the uncapacitated r-allocation p-hub median problem. Comput Oper Res 43:50–60
39. Ruiz R, Stützle T (2006) A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur J Oper Res 177:2033–2049
40. Sánchez Oro J, Mladenović N, Duarte A (2014) General variable neighborhood search for computing graph separators. Optim Lett 1–21. https://doi.org/10.1007/s11590-014-0793-z
41. Sánchez Oro J, Pantrigo JJ, Duarte A (2014) Combining intensification and diversification strategies in VNS. An application to the vertex separation problem. Comput Oper Res 52, Part B(0):209–219. Recent advances in variable neighborhood search
42. Talbi EG (2009) Metaheuristics: from design to implementation. Wiley, Hoboken
43. Todosijević R, Urošević D, Mladenović N, Hanafi S (2015) A general variable neighborhood search for solving the uncapacitated r-allocation p-hub median problem. Optim Lett. https://doi.org/10.1007/s11590-015-0867-6
44. Wu Q, Hao JK, Glover F (2012) Multi-neighborhood tabu search for the maximum weight clique problem. Ann Oper Res 196(1):611–634