Chapter 3
Variable Neighborhood Search

Pierre Hansen, Nenad Mladenović, Jack Brimberg, and José A. Moreno Pérez

Abstract Variable neighborhood search (VNS) is a metaheuristic for solving combinatorial and global optimization problems whose basic idea is a systematic change of neighborhood both within a descent phase to find a local optimum and in a perturbation phase to get out of the corresponding valley. In this chapter we present the basic schemes of VNS and some of its extensions. We then describe recent developments, i.e., formulation space search and variable formulation search. We then present some families of applications in which VNS has proven to be very successful: (1) exact solution of large scale location problems by primal-dual VNS; (2) generation of solutions to large mixed integer linear programs, by hybridization of VNS and local branching; (3) generation of solutions to very large mixed integer programs using VNS decomposition and exact solvers (4) generation of good

P. Hansen
École des Hautes Études Commerciales, Montréal, QC, Canada

GERAD, Montréal, QC, Canada
e-mail: pierre.hansen@gerad.ca

N. Mladenović (✉)
Mathematical Institute, SANU, Belgrade, Serbia
e-mail: nenad@mi.sanu.ac.rs

J. Brimberg
Department of Mathematics and Computer Science, Royal Military College of Canada, Kingston, ON, Canada
e-mail: jack.brimberg@rmc.ca

J. A. M. Pérez
IUDR and Department of Informatics and Systems Engineering, Universidad de La Laguna, Tenerife, Spain
e-mail: jamoreno@ull.es

M. Gendreau, J.-Y. Potvin (eds.), *Handbook of Metaheuristics*,
International Series in Operations Research & Management Science 272,
https://doi.org/10.1007/978-3-319-91086-4_3

feasible solutions to continuous nonlinear programs; (5) adaptation of VNS for solving automatic programming problems from the Artificial Intelligence field and (6) exploration of graph theory to find conjectures, refutations and proofs or ideas of proofs.

3.1 Introduction

Optimization tools have greatly improved during the last two decades. This is due to several factors: (1) progress in mathematical programming theory and algorithmic design; (2) rapid improvement in computer performances; (3) better communication of new ideas and integration in widely used complex softwares. Consequently, many problems long viewed as out of reach are currently solved, sometimes in very moderate computing times. This success, however, has led researchers and practitioners to address much larger instances and more difficult classes of problems. Many of these may again only be solved heuristically. Therefore thousands of papers describing, evaluating and comparing new heuristics appear each year. Keeping abreast of such a large literature is a challenge. Metaheuristics, or general frameworks for building heuristics, are therefore needed in order to organize the study of heuristics. As evidenced by the Handbook, there are many of them. Some desirable properties of metaheuristics [58, 59, 68] are listed in the concluding section of this chapter.

Variable neighborhood search (VNS) is a metaheuristic proposed by some of the present authors some 20 years ago [80]. Earlier work that motivated this approach can be found in [25, 36, 44, 78]. It is based upon the idea of a systematic change of neighborhood both in a descent phase to find a local optimum and in a perturbation phase to get out of the corresponding valley. Originally designed for approximate solution of combinatorial optimization problems, it was extended to address mixed integer programs, nonlinear programs, and recently mixed integer nonlinear programs. In addition VNS has been used as a tool for automated or computer assisted graph theory. This led to the discovery of over 1500 conjectures in that field and the automated proof of more than half of them. This is to be compared with the unassisted proof of about 400 of these conjectures by many different mathematicians.

Applications are rapidly increasing in number and pertain to many fields: location theory, cluster analysis, scheduling, vehicle routing, network design, lot-sizing, artificial intelligence, engineering, pooling problems, biology, phylogeny, reliability, geometry, telecommunication design, etc. References are too numerous to be listed here, but many of them can be found in [69] and special issues of *IMA Journal of Management Mathematics* [76], *European Journal of Operational Research* [68] and *Journal of Heuristics* [87] that are devoted to VNS.

This chapter is organized as follows. In the next section we present the basic schemes of VNS, i.e., variable neighborhood descent (VND), reduced VNS (RVNS), basic VNS (BVNS) and general VNS (GVNS). Two important extensions are presented in Sect. 3.3: Skewed VNS and Variable neighborhood decomposition

search (VNDS). A further recent development called Formulation Space Search (FSS) is discussed in Sect. 3.4. The remainder of the paper describes applications of VNS to several classes of large scale and complex optimization problems for which it has proven to be particularly successful. Section 3.5 is devoted to primal dual VNS (PD-VNS) and its application to location and clustering problems. Finding feasible solutions to large mixed integer linear programs with VNS is discussed in Sect. 3.6. Section 3.7 addresses ways to apply VNS in continuous global optimization. The more difficult case of solving mixed integer nonlinear programming by VNS is considered in Sect. 3.8. Applying VNS to graph theory *per se* (and not just to particular optimization problems defined on graphs) is discussed in Sect. 3.9. Brief conclusions are drawn in Sect. 3.10.

3.2 Basic Schemes

A deterministic optimization problem may be formulated as

$$\min\{f(x)|x \in X, X \subseteq \mathscr{S}\}, \tag{3.1}$$

where $\mathscr{S}, X, x$ and f denote the *solution space*, the *feasible set*, a *feasible solution* and a real-valued *objective function*, respectively. If $\mathscr{S}$ is a finite but large set, a *combinatorial optimization* problem is defined. If $\mathscr{S} = \mathbb{R}^n$, we refer to *continuous optimization*. A solution $x^* \in X$ is *optimal* if

$$f(x^*) \leq f(x), \forall x \in X.$$

An *exact algorithm* for problem (3.1), if one exists, finds an optimal solution x^*, together with the proof of its optimality, or shows that there is no feasible solution, i.e., $X = \emptyset$, or the solution is unbounded. Moreover, in practice, the time needed to do so should be finite (and not too long). For continuous optimization, it is reasonable to allow for some degree of tolerance, i.e., to stop when sufficient convergence is detected.

Let us denote $\mathscr{N}_k$, $(k = 1, \ldots, k_{max})$, a finite set of pre-selected neighborhood structures, and $\mathscr{N}_k(x)$ the set of solutions in the kth neighborhood of x. Most local search heuristics use only one neighborhood structure, i.e., $k_{max} = 1$. Often successive neighborhoods $\mathscr{N}_k$ are nested and may be induced from one or more metric (or quasi-metric) functions introduced into a solution space $\mathscr{S}$. An *optimal solution* x_{opt} (or global minimum) is a feasible solution where a minimum is reached. We call $x' \in X$ a *local minimum* of (3.1) with respect to $\mathscr{N}_k$ (w.r.t. $\mathscr{N}_k$ for short), if there is no solution $x \in \mathscr{N}_k(x') \subseteq X$ such that $f(x) < f(x')$. Metaheuristics (based on local search procedures) try to continue the search by other means after finding the first local minimum. VNS is based on three simple facts:

Fact 1 *A local minimum w.r.t. one neighborhood structure is not necessarily so for another;*

Fact 2 *A global minimum is a local minimum w.r.t. all possible neighborhood structures;*

Fact 3 *For many problems, local minima w.r.t. one or several $\mathcal{N}_k$ are relatively close to each other.*

This last observation, which is empirical, implies that a local optimum often provides some information about the global one. For instance, there may be several variables sharing the same values in both solutions. Since these variables usually cannot be identified in advance, one should conduct an organized study of the neighborhoods of a local optimum until a better solution is found.

In order to solve (1) by using several neighborhoods, facts 1–3 can be used in three different ways: (1) deterministic; (2) stochastic; (3) both deterministic and stochastic.

We first examine in Algorithm 1 the solution move and neighborhood change function that will be used within a VNS framework. Function `NeighborhoodChange()` compares the incumbent value $f(x)$ with the new value $f(x')$ obtained from the kth neighborhood (line 1). If an improvement is obtained, the incumbent is updated (line 2) and k is returned to its initial value (line 3). Otherwise, the next neighborhood is considered (line 4).

Function `NeighborhoodChange` (x, x', k)
1 **if** $f(x') < f(x)$ **then**
2 $\quad x \leftarrow x'$ // Make a move
3 $\quad k \leftarrow 1$ // Initial neighborhood
else
4 $\quad k \leftarrow k+1$ // Next neighborhood
return x, k

Algorithm 1: Neighborhood change

Below we discuss Variable Neighborhood Descent and Reduced Variable Neighborhood Search and then build upon this to construct the framework for Basic and General Variable Neighborhood Search.

(i) The **Variable Neighborhood Descent** (VND) method (Algorithm 2) performs a change of neighborhoods in a deterministic way. These neighborhoods are denoted as $N_k, k = 1, \ldots, k_{max}$.

Most local search heuristics use one or sometimes two neighborhoods for improving the current solution (i.e., $k_{max} \leq 2$). Note that the final solution should be a local minimum w.r.t. all k_{max} neighborhoods, and thus, a global optimum is more likely to be reached than with a single structure. Beside this *sequential* order of neighborhood structures in VND, one can develop a *nested* strategy. Assume, for example, that $k_{max} = 3$; then a possible nested strategy is: perform VND with Algorithm 2 for the first two neighborhoods from each point x' that belongs to the third one ($x' \in N_3(x)$). Such an approach is successfully applied in [22, 26, 57].

Function VND (x, k_{max})
1 $k \leftarrow 1$
2 **repeat**
3 $x' \leftarrow arg\min_{y \in N_k(x)} f(y)$ // Find the best neighbor in $N_k(x)$
4 $x, k \leftarrow$ NeighborhoodChange (x, x', k) // Change neighborhood
until $k = k_{max}$
return x

Algorithm 2: Variable neighborhood descent

(ii) The **Reduced VNS** (RVNS) method is obtained when a random point is selected from $\mathcal{N}_k(x)$ and no descent is attempted from this point. Rather, the value of the new point is compared with that of the incumbent and an update takes place in the case of improvement. We also assume that a stopping condition has been chosen such as the maximum CPU time allowed t_{max}, or the maximum number of iterations between two improvements. To simplify the description of the algorithms, we always use t_{max} below. Therefore, RVNS (Algorithm 3) uses two parameters: t_{max} and k_{max}.

Function RVNS(x, k_{max}, t_{max})
1 **repeat**
2 $k \leftarrow 1$
3 **repeat**
4 $x' \leftarrow$ Shake(x, k)
5 $x, k \leftarrow$ NeighborhoodChange (x, x', k)
until $k = k_{max}$
6 $t \leftarrow$ CpuTime ()
until $t > t_{max}$
return x

Algorithm 3: Reduced VNS

The function Shake in line 4 generates a point x' at random from the kth neighborhood of x, i.e., $x' \in \mathcal{N}_k(x)$. It is given in Algorithm 4, where it is assumed that the points from $\mathcal{N}_k(x)$ are numbered as $\{x^1, \ldots, x^{|\mathcal{N}_k(x)|}\}$. Note that a different notation is used for the neighborhood structures in the shake operation, since these are generally different than the ones used in VND.

Function Shake(x, k)
1 $w \leftarrow \lfloor 1 + \text{Rand}(0,1) \times |\mathcal{N}_k(x)| \rfloor$
2 $x' \leftarrow x^w$
return x'

Algorithm 4: Shaking function

RVNS is useful for very large instances for which local search is costly. It can be used as well for finding initial solutions for large problems before decomposition.

It has been observed that the best value for the parameter k_{max} is often 2 or 3. In addition, a maximum number of iterations between two improvements is typically used as the stopping condition. RVNS is akin to a Monte-Carlo method, but is more systematic (see, e.g., [81] where results obtained by RVNS were 30% better than those of the Monte-Carlo method in solving a continuous min-max problem). When applied to the p-Median problem, RVNS gave equally good solutions as the *Fast Interchange* heuristic of [102] while being 20 to 40 times faster [63].

(iii) The **Basic VNS** (BVNS) method [80] combines deterministic and stochastic changes of neighborhood. The deterministic part is represented by a local search heuristic. It consists in (1) choosing an initial solution x, (2) finding a direction of descent from x (within a neighborhood $N(x)$) and (3) moving to the minimum of $f(x)$ within $N(x)$ along that direction. If there is no direction of descent, the heuristic stops; otherwise it is iterated. Usually the steepest descent direction, also referred to as *best improvement*, is used. Also see Algorithm 2, where the best improvement is used in each neighborhood of the VND. This is summarized in Algorithm 5, where we assume that an initial solution x is given. The output consists of a local minimum, also denoted by x, and its value.

Function `BestImprovement`(x)
1 **repeat**
2 $\quad x' \leftarrow x$
3 $\quad x \leftarrow arg\min_{y \in N(x')} f(y)$
until $(f(x) \geq f(x'))$
return x

Algorithm 5: Best improvement (steepest descent) heuristic

As *Steepest descent* may be time-consuming, an alternative is to use a *first descent* (or *first improvement*) heuristic. Points $x^i \in N(x)$ are then enumerated systematically and a move is made as soon as a direction for descent is found. This is summarized in Algorithm 6.

Function `FirstImprovement`(x)
1 **repeat**
2 $\quad x' \leftarrow x;\ \ i \leftarrow 0$
3 $\quad$ **repeat**
4 $\quad\quad i \leftarrow i+1$
5 $\quad\quad x \leftarrow arg\min\{f(x), f(x^i)\},\ \ x^i \in N(x)$
$\quad$ **until** $(f(x) < f(x')$ `or` $i = |N(x)|)$
until $(f(x) \geq f(x'))$
return x

Algorithm 6: First improvement (first descent) heuristic

The stochastic phase of BVNS (see Algorithm 7) is represented by the random selection of a point x' from the kth neighborhood of the shake operation. Note that

point x' is generated at random in Step 5 in order to avoid cycling, which might occur with a deterministic rule.

```
Function BVNS(x, k_max, t_max)
 1  t ← 0
 2  while t < t_max do
 3  |  k ← 1
 4  |  repeat
 5  |  |  x' ← Shake(x,k)                    // Shaking
 6  |  |  x'' ← BestImprovement(x')          // Local search
 7  |  |  x,k ← NeighborhoodChange(x,x'',k)  // Change neighborhood
    |  until k = k_max
 8  |  t ← CpuTime()
 return x
```

Algorithm 7: Basic VNS

Example. We illustrate the basic steps on a minimum k-cardinality tree instance taken from [72], see Fig. 3.1. The minimum k-cardinality tree problem on graph G (k-card for short) consists of finding a subtree of G with exactly k edges whose sum of weights is minimum.

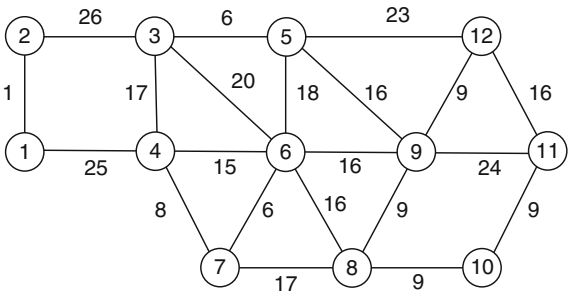

Fig. 3.1 4-Cardinality tree problem

The steps of BVNS for solving the 4-card problem are illustrated in Fig. 3.2. In Step 0 the objective function value, i.e., the sum of edge weights, is equal to 40; it is indicated in the right bottom corner of the figure. That first solution is a local minimum with respect to the edge-exchange neighborhood structure (one edge in, one out). After shaking, the objective function is 60, and after another local search, we are back to the same solution. Then, in Step 3, we take out 2 edges and add another 2 at random, and after a local search, an improved solution is obtained with a value of 39. Continuing in that way, the optimal solution with an objective function value equal to 36 is obtained in Step 8.

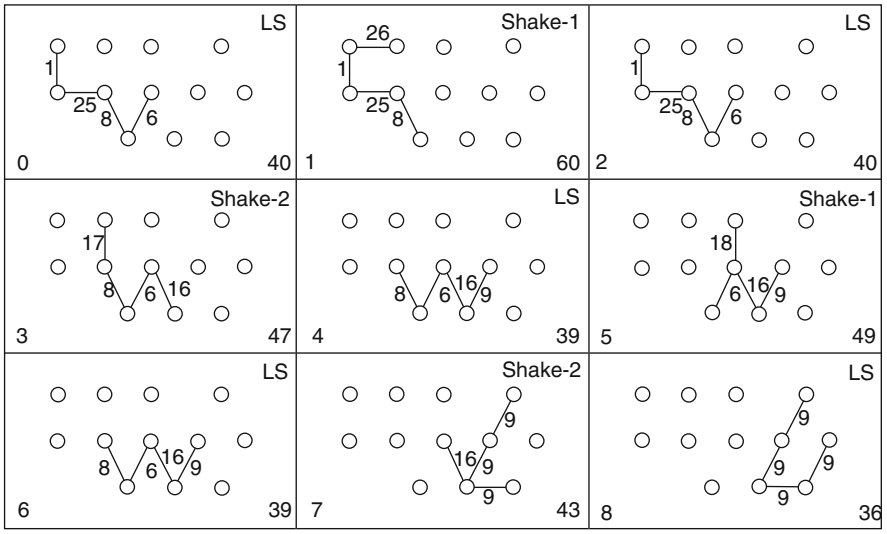

Fig. 3.2 Steps of the Basic VNS for solving 4-card tree problem

(iv) **General VNS.** Note that the local search step (line 6 in BVNS, Algorithm 7) may also be replaced by VND (Algorithm 2). This General VNS (VNS/VND) approach has led to some of the most successful applications reported in the literature (see, e.g., [1, 26–29, 31, 32, 39, 57, 66, 92, 93]). General VNS (GVNS) is outlined in Algorithm 8 below. Note that neighborhoods $N_1, \ldots, N_{l_{max}}$ are used in the VND step, while a different series of neighborhoods $N_1, \ldots, N_{k_{max}}$ apply to the Shake step.

Function GVNS $(x, \ell_{max}, k_{max}, t_{max})$
1 **repeat**
2 $k \leftarrow 1$
3 **repeat**
4 $x' \leftarrow$ Shake(x,k)
5 $x'' \leftarrow$ VND(x', ℓ_{max})
6 $x,k \leftarrow$ NeighborhoodChange(x,x'',k)
until $k = k_{max}$
7 $t \leftarrow$ CpuTime()
until $t > t_{max}$
return x

Algorithm 8: General VNS

3.3 Some Extensions

(i) The **Skewed VNS** (SVNS) method [62] addresses the problem of exploring valleys far from the incumbent solution. Indeed, once the best solution in a large region has been found it is necessary to go quite far to obtain an improved one. Solutions drawn at random in far-away neighborhoods may differ substantially from the incumbent, and VNS may then degenerate, to some extent, into a Multistart heuristic (where descents are made iteratively from solutions generated at random, and which is known to be inefficient). So some compensation for distance from the incumbent must be made, and a scheme called Skewed VNS (SVNS) is proposed for that purpose. Its steps are presented in Algorithms 9, 10 and 11. The `KeepBest`(x,x') function (Algorithm 9) in SVNS simply keeps the best of solutions x and x' The `NeighborhoodChangeS` function (Algorithm 10) performs the move and neighborhood change for the SVNS.

Function `KeepBest`(x,x')
if $f(x') < f(x)$ **then**
  $x \leftarrow x'$
return x

Algorithm 9: Keep best solution

Function `NeighborhoodChangeS`(x,x',k,α)
if $f(x') - \alpha\rho(x,x') < f(x)$ **then**
  $x \leftarrow x'$; $k \leftarrow 1$
else
  $k \leftarrow k+1$
return x,k

Algorithm 10: Neighborhood change for Skewed VNS

SVNS makes use of a function $\rho(x,x'')$ to measure the distance between the current solution x and the local optimum x''. The distance function used to define $\mathcal{N}_k$ could also be used for this purpose. The parameter α must be chosen to allow movement to valleys far away from x when $f(x'')$ is larger than $f(x)$ but not too much larger (otherwise one will always leave x). A good value for α is found experimentally in each case. Moreover, in order to avoid frequent moves from x to a close solution, one may take a smaller value for α when $\rho(x,x'')$ is small. More sophisticated choices for selecting a function of $\alpha\rho(x,x'')$ could be made through some learning process.

Function SVNS $(x, k_{max}, t_{max}, \alpha)$
1 $x_{best} \leftarrow x$
2 **repeat**
3 $k \leftarrow 1$
4 **repeat**
5 $x' \leftarrow$ Shake(x,k)
6 $x'' \leftarrow$ FirstImprovement(x')
7 $x,k \leftarrow$ NeighborhoodChangeS(x,x'',k,α)
8 $x_{best} \leftarrow$ KeepBest (x_{best},x)
until $k = k_{max}$
9 $x \leftarrow x_{best}$
10 $t \leftarrow$ CpuTime()
until $t > t_{max}$
return x

Algorithm 11: Skewed VNS

(ii) The **Variable neighborhood decomposition search** (VNDS) method [63] extends the basic VNS into a two-level VNS scheme based upon decomposition of the problem. It is presented in Algorithm 12, where t_d is an additional parameter that represents the running time allowed for solving decomposed (smaller-sized) problems by Basic VNS (line 5).

Function VNDS $(x, k_{max1}, t_{max}, t_d)$
1 **repeat**
2 $k \leftarrow 1$
3 **repeat**
4 $x' \leftarrow$ Shake (x,k); $y \leftarrow x' \setminus x$
5 $y' \leftarrow$ BVNS(y, k_{max2}, t_d); $x'' = (x' \setminus y) \cup y'$
6 $x''' \leftarrow$ FirstImprovement(x'')
7 $x,k \leftarrow$ NeighborhoodChange(x,x''',k)
until $k = k_{max1}$
until $t > t_{max}$
return x

Algorithm 12: Variable neighborhood decomposition search

For ease of presentation, but without loss of generality, we assume that the solution x represents a set of attributes. In Step 4 we denote by y a set of k solution attributes present in x' but not in x ($y = x' \setminus x$). In Step 5 we find the local optimum y' in the space of y; then we denote with x'' the corresponding solution in the whole space X ($x'' = (x' \setminus y) \cup y'$). We notice that exploiting some *boundary effects* in a new solution can significantly improve solution quality. That is why, in Step 6, the local optimum x''' is found in the whole space X using x'' as an initial solution. If this is time consuming, then at least a few local search iterations should be performed.

VNDS can be viewed as embedding the classical successive approximation scheme (which has been used in combinatorial optimization at least since the sixties, see, e.g., [48]) in the VNS framework. Let us mention here a few applications

of VNDS: p-median problem [63]; simple plant location problem [67]; k-cardinality tree problem [100]; 0-1 mixed integer programming problem [51, 74]; design of MBA student teams [37], etc.

3.4 Changing Formulation Within VNS

A traditional approach to tackle an optimization problem is to consider a given formulation and search in some way through its feasible set X. Given that the same problem can often be formulated in different ways, it is possible to extend search paradigms to include jumps from one formulation to another. Each formulation should lend itself to some traditional search method, its 'local search' that works totally within this formulation, and yields a final solution when started from some initial solution. Any solution found in one formulation should easily be translatable to its equivalent solution in any other formulation. We may then move from one formulation to another by using the solution resulting from the local search of the former as an initial solution for the local search of the latter. Such a strategy will of course only be useful when local searches in different formulations behave differently. Here we discuss two such possibilities.

3.4.1 Variable Neighborhood-Based Formulation Space Search

The idea of changing the formulation of a problem was investigated in [82, 83] using an approach that systematically alternates between different formulations for solving various Circle Packing Problems (CPP). It is shown there that a stationary point for a nonlinear programming formulation of CPP in Cartesian coordinates is not necessarily a stationary point in polar coordinates. A method called *Reformulation Descent* (RD) that alternates between these two formulations until the final solution is stationary with respect to both formulations is suggested. Results obtained were comparable with the best known values, but were achieved about 150 times faster than with an alternative single formulation approach. In this paper, the idea suggested above of *Formulation Space Search* (FSS) is also introduced, using more than two formulations. Some research in that direction has also been reported in [70, 79, 90]. One methodology that uses the variable neighborhood idea when searching through the formulation space is given in Algorithms 13 and 14. Here ϕ (ϕ') denotes a formulation from a given space $\mathscr{F}$, x (x') denotes a solution in the feasible set defined with that formulation, and $\ell \leq \ell_{max}$ is the formulation neighborhood index. Note that Algorithm 14 uses a reduced VNS strategy in the formulation space $\mathscr{F}$. Note also that the `ShakeFormulation()` function must provide a search through the solution space $\mathscr{S}'$ (associated with formulation ϕ') in order to get a new solution x'. Any appropriate method can be used for this purpose.

Function `FormulationChange`(x,x',ϕ,ϕ',ℓ)
1 **if** $f(\phi',x') < f(\phi,x)$ **then**
2 $\quad \phi \leftarrow \phi'$
3 $\quad x \leftarrow x'$
4 $\quad \ell \leftarrow 1$
else
5 $\quad \ell \leftarrow \ell + 1$
6 **return** x, ϕ, ℓ

Algorithm 13: Formulation change

Function `VNFSS`(x, ϕ, ℓ_{max})
1 **repeat**
2 $\quad \ell \leftarrow 1$ // Initialize formulation in $\mathscr{F}$
3 $\quad$ **while** $\ell \leq \ell_{max}$ **do**
4 $\quad\quad x', \phi', \ell \leftarrow$ `ShakeFormulation`(x,x',ϕ,ϕ',ℓ) // $(\phi',x') \in (N_\ell(\phi), \mathscr{N}(x))$ random
5 $\quad\quad x, \phi, \ell \leftarrow$ `FormulationChange`(x,x',ϕ,ϕ',ℓ) // Change formulation
until *some stopping condition is met*
6 **return** x

Algorithm 14: Reduced variable neighborhood FSS

3.4.2 Variable Formulation Search

Many optimization problems in the literature, e.g., min-max problems, demonstrate a flat landscape. It means that, given a formulation of the problem, many neighbors of a solution have the same objective function value. When this happens, it is difficult to determine which neighborhood solution is more promising to continue the search. To address this drawback, the use of alternative formulations of the problem within VNS is proposed in [85, 86, 89]. In [89] it is named Variable Formulation Search (VFS). It combines a change of neighborhood within the VNS framework, with the use of alternative formulations.

Let us assume that, beside the original formulation and the corresponding objective function $f_0(x)$, there are p other formulations denoted as $f_1(x), .., f_p(x), x \in X$. Note that two formulations are defined as equivalent if the optimal solution of one is the optimal solution of the other, and vice versa. For simplification purposes, we will denote different formulations as different objectives $f_i(x), i = 1, .., p$. The idea of VFS is to add the procedure `Accept`(x,x',p), given in Algorithm 15, in all three basic steps of BVNS: `Shaking`, `LocalSearch` and `NeighborhoodChange`. Clearly, if a better solution is not obtained by any of the $p+1$ formulations, the move is rejected. The next iteration in the loop of Algorithm 15 will take place only if the objective function values according to all previous formulations are equal.

Logical Function `Accept` (x, x', p)
for $i = 0$ *to* p **do**
  if $(f_i(x') < f_i(x))$ **then return** `TRUE`
  if $(f_i(x') > f_i(x))$ **then return** `FALSE`
return `FALSE`

Algorithm 15: Accept procedure with p secondary formulations

If `Accept` (x, x', p) is included in the `LocalSearch` subroutine of BVNS, then it will not stop the first time a non improved solution is found. In order to stop `LocalSearch` and thus claim that x' is a local minimum, x' should not be improved by any among the p different formulations. Thus, for any particular problem, one needs to design different formulations of the problem considered and decide the order in which they will be used in the `Accept` subroutine. Answers to those two questions are problem specific and sometimes not easy. The `Accept` (x, x', p) subroutine can obviously be added to the `NeighborhoodChange` and `Shaking` steps of BVNS from Algorithm 7 as well.

In [85], three evaluation functions, or acceptance criteria, within the `Neighborhood Change` step are used in solving the *Bandwidth Minimization Problem*. This min-max problem consists of finding permutations of rows and columns of a given square matrix to minimize the maximal distance of the nonzero elements from the main diagonal in the corresponding rows. Solution x may be represented as a labeling of a graph and the move from x to x' as $x \rightarrow x'$. Three criteria are used:

1. the bandwidth length $f_0(x)$ $(f_0(x') < f_0(x))$;
2. the total number of critical vertices $f_1(x)$ $(f_1(x') < f_1(x))$, if $f_0(x') = f_0(x)$;
3. $f_3(x, x') = \rho(x, x') - \alpha$, if $f_0(x') = f_0(x)$ and $f_1(x') = f_1(x)$. Here, we want $f_3(x, x') > 0$, because we assume that x and x' are sufficiently far from one another when $\rho(x, x') > \alpha$, where α is an additional parameter. The idea for a move to an even worse solution, if it is very far, is used within Skewed VNS. However, a move to a solution with the same value is only performed in [85] if its Hamming distance from the incumbent is greater than α.

In [86] a different mathematical programming formulation of the original problem is used as a secondary objective within the `Neighborhood Change` function of VNS. There, two combinatorial optimization problems on a graph are considered: the *Metric Dimension Problem* and *Minimal Doubly Resolving Set Problem*.

A more general VFS approach is given in [89], where the *Cutwidth Graph Minimization Problem* (CWP) is considered. CWP also belongs to the min-max problem family. For a given graph, one needs to find a sequence of nodes such that the maximum cutwidth is minimum. The cutwidth of a graph should be clear from the example provided in Fig. 3.3 for the graph with six vertices and nine edges shown in (a).

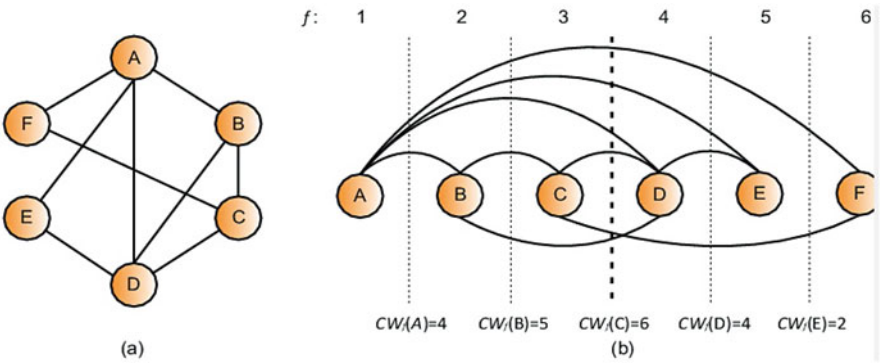

Fig. 3.3 Cutwidth minimization example as in [89]

Figure 3.3b shows an ordering x of the vertices of the graph in (a) with the corresponding cutwidth CW values of each vertex. It is clear that the CW represents the number of cut edges between two consecutive nodes in the solution x. The cutwidth value $f_0(x) = CW(x)$ of the ordering $x = (A,B,C,D,E,F)$ is equal to $f_0(x) = \max\{4,5,6,4,2\} = 6$. Thus, one needs to find an order x that minimizes the maximum cut-width value over all vertices.

Beside minimizing the bandwidth f_0, two additional formulations, denoted f_1 and f_2 , are used in [89], and implemented within a VND local search. Results are compared among themselves (Table 3.1) and with a few heuristics from the literature (Table 3.1), using the following usual data set:

- "*Grid*": This data set consists of 81 matrices constructed as the Cartesian product of two paths. They were originally introduced by Rolim et al. [94]. For this set of instances, the vertices are arranged on a grid of dimension width $\times$ height where width and height are selected from the set {3, 6, 9, 12, 15, 18, 21, 24, 27}.
- "*Harwell-Boeing*" (HB): This data set is a subset of the public-domain Matrix Market library.[1] This collection consists of a set of standard test matrices $M = (M_{ij})$ arising from problems in linear systems, least squares, and eigenvalue calculations from a wide variety of scientific and engineering disciplines. Graphs were derived from these matrices by considering an edge (i, j) for every element $M_{ij} \neq 0$. The data set is formed by the selection of the 87 instances were $n \leq 700$. Their number of vertices ranges from 30 to 700 and the number of edges from 46 to 41,686.

[1] Available at http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/.

Table 3.1 presents the results obtained with four different VFS variants, after executing them for 30 s over each instance. The column 'BVNS' of Table 3.1 represents a heuristic based on BVNS which makes use only of the original formulation f_0 of the CWP. VFS_1 denotes a BVNS heuristic that uses only one secondary criterion, i.e., f_0 and f_1. VFS_2 is equivalent to the previous one with the difference that now f_2 is considered (instead of f_1). Finally, the fourth column of the table, denoted as VFS_3, combines the original formulation of the CWP with the two alternative ones, in the way presented in Algorithm 15. All algorithms were configured with $k_{max} = 0.1n$ and start from the same random solution.

Table 3.1 Comparison of alternative formulations within 30 s for each test, by average objective values and % deviation from the best known solution

	BVNS	VFS_1	VFS_2	VFS_3
Avg.	137.31	93.56	91.56	90.75
Dev. (%)	192.44	60.40	49.23	48.22

Test are performed on "Grid" and "HB" data sets that contain 81 and 86 instances, respectively

It appears that significant improvements in solution quality are obtained when at least one secondary formulation is used in case of ties (compare e.g., 192.44% and 60.40% deviations from the best known solutions obtained by BVNS and VFS_1, respectively). An additional improvement is obtained when all three formulations are used in VFS_3.

Comparison of VFS_3 and state-of-the-art heuristics are given in Table 3.2. There, the stopping condition is increased from 30 s to 300 and 600 s for the first and the second set of instances, respectively. Besides average values and % deviation, the methods are compared based on the number of wins (the third row) and the total cpu time in seconds. Overall, the best quality results are obtained by VFS in less computing time.

Table 3.2 Comparison of VFS with the state-of-the-art heuristics over the "Grid" and "HB" data sets, within 300 and 600 s respectively

	81 'grid' test instances				86 HB instances			
	GPR [2]	SA [34]	SS [88]	VFS [89]	GPR [2]	SA [34]	SS [88]	VFS [89]
Avg.	38.44	16.14	13.00	12.23	364.83	346.21	315.22	314.39
Dev. (%)	201.81	25.42	7.76	3.25	95.13	53.30	3.40	1.77
#Opt.	2	37	44	59	2	8	47	61
CPU t (s)	235.16	216.14	210.07	90.34	557.49	435.40	430.57	128.12

3.5 Primal-Dual VNS

For most modern heuristics, the difference in value between the optimal solution and the obtained approximate solution is not precisely known. Guaranteed performance of the primal heuristic may be determined if a lower bound on the objective

function value can be found. To this end, the standard approach is to relax the integrality condition on the primal variables, based on a mathematical programming formulation of the problem. However, when the dimension of the problem is large, even the relaxed problem may be impossible to solve exactly by standard commercial solvers. Therefore, it seems to be a good idea to solve dual relaxed problems heuristically as well. In this way we get guaranteed bounds on the primal heuristic performance. The next difficulty arises if we want to get an exact solution within a branch-and-bound framework since having the approximate value of the relaxed dual does not allow us to branch in an easy way, for example by exploiting complementary slackness conditions. Thus, the exact value of the dual is necessary. A general approach to get both guaranteed bounds and an exact solution is proposed in [67], and referred as Primal-Dual VNS (PD-VNS). It is given in Algorithm 16.

Function PD-VNS (x, k_{max}, t_{max})
1 BVNS (x, k_{max}, t_{max}) // Solve primal by VNS
2 DualFeasible(x, y) // Find (infeasible) dual such that $f_P = f_D$
3 DualVNS(y) // Use VNS do decrease infeasibility
4 DualExact(y) // Find exact (relaxed) dual
5 BandB(x, y) // Apply branch-and-bound method

Algorithm 16: Basic PD-VNS

In the first stage, a heuristic procedure based on VNS is used to obtain a near optimal solution. In [67] it is shown that VNS with decomposition is a very powerful technique for large-scale simple plant location problems (SPLP) with up to 15,000 facilities and 15,000 users. In the second phase, the objective is to find an exact solution of the relaxed dual problem. Solving the relaxed dual is accomplished in three stages: (1) find an initial dual solution (generally infeasible) using the primal heuristic solution and complementary slackness conditions; (2) find a feasible solution by applying VNS to the unconstrained nonlinear form of the dual; (3) solve the dual exactly starting with the found initial feasible solution using a customized "sliding simplex" algorithm that applies "windows" on the dual variables, thus substantially reducing the problem size. On all problems tested, including instances much larger than those previously reported in the literature, the procedure was able to find the exact dual solution in reasonable computing time. In the third and final phase, armed with tight upper and lower bounds obtained from the heuristic primal solution in phase one and the exact dual solution in phase two, respectively, a standard branch-and-bound algorithm is applied to find an optimal solution of the original problem. The lower bounds are updated with the dual sliding simplex method and the upper bounds whenever new integer solutions are obtained at the nodes of the branching tree. In this way it was possible to solve exactly problem instances of sizes up to 7000 facilities $\times$7000 users, for uniform fixed costs, and $15,000$ facilities $\times 15,000$ users, otherwise.

3.6 VNS for Mixed Integer Linear Programming

The Mixed Integer Linear Programming (MILP) problem consists of maximizing or minimizing a linear function, subject to equality or inequality constraints and integrality restrictions on some of the variables. The mixed integer programming problem $(MILP)$ can be expressed as:

$$(MILP) \quad \begin{bmatrix} \min & \sum_{j=1}^{n} c_j x_j & \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j \geq b_i & \forall i \in M = \{1,2,\ldots,m\} \\ & x_j \in \{0,1\} & \forall j \in \mathscr{B} \\ & x_j \geq 0, \texttt{integer} & \forall j \in \mathscr{G} \\ & x_j \geq 0 & \forall j \in \mathscr{C} \end{bmatrix}$$

where the set of indices $N = \{1,2,\ldots,n\}$ is partitioned into three subsets $\mathscr{B}, \mathscr{G}$ and $\mathscr{C}$, corresponding to binary, general integer and continuous variables, respectively.

Numerous combinatorial optimization problems, including a wide range of practical problems in business, engineering and science, can be modeled as MILPs. Several special cases, such as knapsack, set packing, cutting and packing, network design, protein alignment, traveling salesman and other routing problems, are known to be NP-hard [46].

Many commercial solvers such as CPLEX [71] are available for solving MILPs. Methods included in such software packages are usually of the branch-and-bound (B&B) or of branch-and-cut (B&C) types. Basically, those methods enumerate all possible integer values in some order, and prune the search space for the cases where such enumeration cannot improve the current best solution.

3.6.1 Variable Neighborhood Branching

The connection between local search based heuristics and exact solvers may be established by introducing the so called *local branching constraints* [43]. By adding just one constraint into (MILP), as explained below, the kth neighborhood of (MILP) is defined. This allows the use of all local search based metaheuristics, such as Tabu search, Simulating annealing, VNS etc. More precisely, given two solutions x and y of (MILP), the distance between x and y is defined as:

$$\delta(x,y) = \sum_{j \in \mathscr{B}} | x_j - y_j |.$$

Let X be the solution space of (MILP). The neighborhood structures $\{\mathcal{N}_k \mid k = 1,\ldots,k_{max}\}$ can be defined, knowing the distance $\delta(x,y)$ between any two solutions $x,y \in X$. The set of all solutions in the kth neighborhood of $y \in X$ is denoted as $\mathcal{N}_k(y)$ where

$$\mathcal{N}_k(y) = \{x \in X \mid \delta(x,y) \leq k\}.$$

For the pure 0-1 MILP given above (i.e., (MILP) with $\mathcal{G} = \emptyset$), $\delta(.,.)$ represents the Hamming distance and $\mathcal{N}_k(y)$ may be expressed by the following *local branching constraint*

$$\delta(x,y) = \sum_{j \in S}(1 - x_j) + \sum_{j \in \mathcal{B} \setminus S} x_j \leq k, \tag{3.2}$$

where $S = \{j \in \mathcal{B} \mid y_j = 1\}$.

In [66] a general VNS procedure for solving 0-1 MILPs is presented (see Algorithm 17). An exact MILP solver (`MIPSOLVE()` within CPLEX) is used as a black box for finding the best solution in the neighborhood, based on the given formulation (MILP) plus the added local branching constraints. Shaking is performed using the Hamming distance defined above. A detailed description of this VNS branching method is provided in Algorithm 17. The variables and constants used in the algorithm are defined as follows [66]:

- UB—input variable for the CPLEX solver which represents the current upper bound.
- *first*—logical input variable for CPLEX solver which is `true` if the first solution lower than UB is asked for in the output; if *first* = `false`, CPLEX returns the best solution found so far.
- TL—maximum time allowed for running CPLEX.
- *rhs*—right hand side of the local branching constraint; it defines the size of the neighborhood within the inner or VND loop.
- *cont*—logical variable which indicates if the inner loop continues (`true`) or not (`false`).
- *x_opt* and *f_opt*—incumbent solution and corresponding objective function value.
- *x_cur*, *f_cur*, *k_cur*—current solution, objective function value and neighborhood from where the VND local search starts (lines 6–20).
- *x_next* and *f_next*—solution and corresponding objective function value obtained by CPLEX in the inner loop.

```
Function VnsBra(total_time_limit, node_time_limit, k_step,
    x_opt)
 1  TL := total_time_limit; UB := ∞; first := true
 2  stat := MIPSOLVE(TL, UB, first, x_opt, f_opt)
 3  x_cur:=x_opt; f_cur:=f_opt
 4  while (elapsedtime < total_time_limit) do
 5      cont := true; rhs := 1; first := false
 6      while (cont or elapsedtime < total_time_limit) do
 7          TL = min(node_time_limit, total_time_limit-
            elapsedtime)
 8          add local br. constr. δ(x,x_cur) ≤ rhs; UB := f_cur
 9          stat := MIPSOLVE(TL, UB, first, x_next, f_next)
10          switch stat do
11              case "opt_sol_found":
12                  reverse last local br. const. into δ(x,x_cur) ≥ rhs+1
13                  x_cur := x_next; f_cur := f_next; rhs := 1;
14              case "feasible_sol_found":
15                  reverse last local br. constr. into δ(x,x_cur) ≥ 1
16                  x_cur := x_next; f_cur := f_next; rhs := 1;
17              case "proven_infeasible":
18                  remove last local br. constr.; rhs := rhs+1;
19              case "no_feasible_sol_found":
20                  cont := false
21      if f_cur < f_opt then
22          x_opt := x_cur; f_opt := f_cur; k_cur := k_step;
        else
23          k_cur := k_cur+k_step;
24      remove all added constraints; cont := true
25      while cont and (elapsedtime < total_time_limit) do
26          add constraints k_cur ≤ δ(x,x_opt) and
            δ(x,x_opt) < k_cur+k_step
27          TL := total_time_limit-elapsedtime; UB := ∞; first := true
28          stat := MIPSOLVE(TL, UB, first, x_cur, f_cur)
29          remove last two added constraints; cont =false
30          if stat = "proven_infeasible" or
            "no_feasible_sol_found" then
31              cont :=true; k_cur := k_cur+k_step
```

Algorithm 17: VNS branching

In line 2, a commercial MIP solver is run to get an initial feasible solution, i.e., logical variable 'first' is set to value `true`. The outer loop starts from line 4. VND based local search is performed in the inner loop that starts from line 6 and finishes

at line 24. There are four different outputs from subroutine `MIPSOLVE` provided by variable *stat*. They are coded in lines 11–20. The shaking step also uses the MIP solver. It is presented in the loop that starts at line 25.

3.6.2 VNDS Based Heuristics for MILP

It is well known that heuristics and relaxations are useful for providing upper and lower bounds on the optimal value of large and difficult optimization problems. A hybrid approach for solving 0-1 MILPs is presented in this section. A more detailed description may be found in [51]. It combines variable neighborhood decomposition search (VNDS) [63] and a generic MILP solver for upper bounding purposes, and a generic linear programming solver for lower bounding. VNDS is used to define a variable fixing scheme for generating a sequence of smaller subproblems, which are normally easier to solve than the original problem. Different heuristics are derived by choosing different strategies for updating lower and upper bounds, and thus defining different schemes for generating a series of subproblems. We also present in this section a two-level decomposition scheme, in which subproblems created according to the VNDS rules are further divided into smaller subproblems using another criterion, derived from the mathematical formulation of the problem.

3.6.2.1 VNDS for 0-1 MILPs with Pseudo-Cuts

Variable neighborhood decomposition search is a two-level variable neighborhood search scheme for solving optimization problems, based upon the decomposition of the problem (see Algorithm 12). We discuss here an algorithm which solves exactly a sequence of reduced problems obtained from a sequence of linear programming relaxations. The set of reduced problems for each LP relaxation is generated by fixing a certain number of variables according to VNDS rules. That way, two sequences of upper and lower bounds are generated, until an optimal solution of the problem is obtained. Also, after each reduced problem is solved, a pseudo-cut is added to guarantee that this subproblem is not revisited. Furthermore, whenever an improvement in the objective function value occurs, a local search procedure is applied in the whole solution space to attempt a further improvement (the so-called *boundary effect* within VNDS). This procedure is referred to as VNDS-PC, since it employs VNDS to solve 0-1 MILPs, while incorporating pseudo-cuts to reduce the search space [51].

If $J \subseteq \mathscr{B}$, we define the partial distance between x and y, relative to J, as $\delta(J,x,y) = \sum_{j\in J} | x_j - y_j |$. Obviously we have $\delta(\mathscr{B},x,y) = \delta(x,y)$). More generally, let $\bar{x}$ be an optimal solution of LP(P), the LP relaxation of the problem P considered (not necessarily MIP feasible), and $J \subseteq B(\bar{x}) = \{j \in N \mid \bar{x}_j \in \{0,1\}\}$ an arbitrary subset of indices. The partial distance $\delta(J,x,\bar{x})$ can be linearized as follows:

$$\delta(J,x,\bar{x}) = \sum_{j\in J}[x_j(1-\bar{x}_j)+\bar{x}_j(1-x_j)].$$

Let X be the solution space of problem P. The neighborhood structures $\{\mathscr{N}_k \mid k = k_{min},\ldots,k_{max}\}$, $1 \le k_{min} \le k_{max} \le p$, can be defined knowing the distance $\delta(\mathscr{B},x,y)$

between any two solutions $x, y \in X$. The set of all solutions in the kth neighborhood of $x \in X$ is denoted as $\mathscr{N}_k(x)$, where

$$\mathscr{N}_k(x) = \{y \in X \mid \delta(\mathscr{B}, x, y) \leq k\}.$$

From the definition of $\mathscr{N}_k(x)$, it follows that $\mathscr{N}_k(x) \subset \mathscr{N}_{k+1}(x)$, for any $k \in \{k_{min}, k_{min}+1, \ldots, k_{max}-1\}$, since $\delta(\mathscr{B}, x, y) \leq k$ implies $\delta(\mathscr{B}, x, y) \leq k+1$. It is trivial that, if we completely explore neighborhood $\mathscr{N}_{k+1}(x)$, it is not necessary to explore neighborhood $\mathscr{N}_k(x)$.

Ordering variables w.r.t LP–relaxation. The first variant of VNDS-PC, denoted as `VNDS-PC1`, is considered here for the maximization case. See Algorithm 18 for the pseudo-code of this algorithm which can be easily adjusted for minimization problems. Input parameters for the algorithm are an instance P of the 0-1 MIP problem, a parameter d which defines the number of variables to be released in each iteration and an initial feasible solution x^* of P. The algorithm returns the best solution found until the stopping criterion defined by the variable *proceed*1 is met.

Function `VNDS-PC1`(P, d, x^*)
1 Choose stopping criteria (set *proceed*1 = *proceed*2 = `true`)
2 Add objective cut: $LB = c^{\mathrm{T}}x^*$; $P = (P \mid c^{\mathrm{T}}x > LB)$
3 **while** *proceed*1 **do**
4 Find an optimal solution $\bar{x}$ of LP(P)
5 set $UB = c^{\mathrm{T}}\bar{x}$
6 **if** $B(\bar{x}) = \mathscr{B}$ **then**
break
7 Set $\delta_j = \mid x_j^* - \bar{x}_j \mid$, $j \in \mathscr{B}$
8 Index x_j so that $\delta_j \leq \delta_{j+1}$, $j = 1, \ldots, p-1$, $p = |\mathscr{B}|$
9 Set $q = \mid \{j \in \mathscr{B} \mid \delta_j \neq 0\} \mid$
10 Set $k_{min} = p - q$, $k_{step} = \lfloor q/d \rfloor$, $k_{max} = p - k_{step}$, $k = k_{max}$
11 **while** *proceed*2 **and** $k \geq 0$ **do**
12 $J_k = \{1, \ldots, k\}$; $x' =$ `MIPSOLVE`$(P(x^*, J_k), x^*)$
13 $P = (P \mid \delta(J_k, x^*, x) \geq 1)$
14 **if** $(c^{\mathrm{T}}x' > c^{\mathrm{T}}x^*)$ **then**
15 $x^* =$ `LocalSearch`(P, x'); $LB = c^{\mathrm{T}}x^*$
16 Update objective cut: $P = (P \mid c^{\mathrm{T}}x > LB)$; **break**
else
17 **if** $(k - k_{step} < k_{min})$ **then**
$k_{step} = \max\{\lfloor k/2 \rfloor, 1\}$
18 Set $k = k - k_{step}$
19 Update *proceed*2
20 Update *proceed*1
return LB, UB, x^*.

Algorithm 18: VNDS for MIPs with pseudo-cuts

This variant of VNDS-PC is based on the following choices. Variables are ordered according to their distances from the corresponding LP relaxation solution values (see lines 4, 6 and 7 in Algorithm 18). More precisely, we compute distances $\delta_j = \mid x_j - \bar{x}_j \mid$ for $j \in \mathscr{B}$, where x_j is a variable value of the current incumbent (feasible) solution and $\bar{x}_j$ a variable value of the LP-relaxation. We then index variables $x_j, j \in \mathscr{B}$, so that $\delta_1 \leq \delta_2 \leq \ldots \leq \delta_p$, $p = \mid \mathscr{B} \mid$. Parameters k_{min}, k_{step} and k_{max} (see line 9 in Algorithm 18) are determined in the following way. Let q be the number of binary variables which have different values in the LP relaxation solution and in the incumbent solution ($q = \mid \{j \in \mathscr{B} \mid \delta_j \neq 0\} \mid$), and let d be a given parameter (whose value is experimentally found) which controls the neighborhood size. Then we set $k_{min} = p - q$, $k_{step} = \lfloor q/d \rfloor$ and $k_{max} = p - k_{step}$. We also allow the value of k to be less then k_{min} (see lines 17 and 18 in Algorithm 18). In other words, we allow the variables which have the same integer value in the incumbent and LP-relaxation solutions to be freed anyway. When $k < k_{min}$, k_{step} is set to (approximately) half the number of the remaining fixed variables. Note that the maximum value of parameter k (which is k_{max}) indicates the maximum possible number of fixed variables, which implies the minimum number of free variables and therefore the minimum possible neighborhood size in the VNDS scheme.

If an improvement occurs after solving the subproblem $P(x^*, J_k)$, where x^* is the current incumbent solution (see line 12 in Algorithm 18), we perform a local search on the complete solution, starting from x' (see line 14 in Algorithm 18). The local search applied at this stage is the variable neighborhood descent for 0-1 MILPs, as described in [66]. Note that, in Algorithm 18 and in the pseudo-codes that follow, the statement $y = \texttt{MILPSOLVE}(P, x)$ denotes a call to a generic MILP solver, for a given 0-1 MILP problem P, starting from a given solution x and returning a new solution y (if P is infeasible, then the value of y remains the same as the one before the call to the MILP solver).

In practice, when used as a heuristic with a time limit as the stopping criterion, VNDS-PC1 has a good performance. One can observe that, if pseudo-cuts (line 13 in Algorithm 18) and objective cuts (lines 2 and 16) are not added, the algorithm from [74] is obtained, which is a special case of VNDS-PC with a fixed LP relaxation reference solution.

Ordering variables w.r.t. the minimum and maximum distances from the incumbent solution.

In the VNDS variant above, the variables in the incumbent integer solution are ordered according to the distances of their values to the values of the current linear relaxation solution. However, it is possible to employ different ordering strategies. For example, in the case of maximization of $c^T x$, consider the following two problems:

$$(LP^{-}_{x^*}) \begin{bmatrix} \min \delta(x^*, x) \\ \text{s.t.: } Ax \leq b \\ c^{\mathrm{T}}x \geq LB + 1 \\ x_j \in [0,1],\ j \in \mathscr{B} \\ x_j \geq 0,\ j \in N \end{bmatrix} \qquad (LP^{+}_{x^*}) \begin{bmatrix} \max \delta(x^*, x) \\ \text{s.t.: } Ax \leq b \\ c^{\mathrm{T}}x \geq LB + 1 \\ x_j \in [0,1],\ j \in \mathscr{B} \\ x_j \geq 0,\ j \in N \end{bmatrix}$$

where x^* is the best known integer feasible solution and LB is the best lower bound found so far (i.e., $LB = c^{\mathrm{T}}x^*$). Of course, in case of solving $\min c^T x$, the inequality $c^T x \geq LB + 1$ from models $(LP^{-}_{x^*})$ and $(LP^{+}_{x^*})$, should be replaced with $c^T x \leq UB - 1$, where the upper bound $UB = c^T x^*$. If $\overline{x}^-$ and $\overline{x}^+$ are optimal solutions of the LP-relaxation problems $LP^{-}_{x^*}$ and $LP^{+}_{x^*}$, respectively, then components of x^* could be ordered in ascending order of values $|\overline{x}^-_j - \overline{x}^+_j|$, $j \in \mathscr{B}$. Since both solution vectors $\overline{x}^-$ and $\overline{x}^+$ are real-valued (i.e., from $\mathbb{R}^n$), this ordering technique is expected to be more sensitive than the standard one, i.e., the number of pairs (j, j'), $j, j' \in N, j \neq j'$ for which $|\overline{x}^-_j - \overline{x}^+_j| \neq |\overline{x}^-_{j'} - \overline{x}^+_{j'}|$ is expected to be greater than the number of pairs (h, h'), $h, h' \in N, h \neq h'$ for which $|x^*_h - \overline{x}_h| \neq |x^*_{h'} - \overline{x}_{h'}|$, where $\overline{x}$ is an optimal solution of the LP relaxation $LP(P)$.

Also, according to the definition of $\overline{x}^-$ and $\overline{x}^+$, it is intuitively more likely for the variables x_j, $j \in N$, for which $\overline{x}^-_j = \overline{x}^+_j$, to have that same value $\overline{x}^-_j$ in the final solution, than it is for variables x_j, $j \in N$, for which $x^*_j = \overline{x}_j$ (and $\overline{x}^-_j \neq \overline{x}^+_j$), to have the final value x^*_j. In practice, if $\overline{x}^-_j = \overline{x}^+_j$, $j \in N$, then usually $x^*_j = \overline{x}^-_j$, which justifies the ordering of components of x^* in the described way. However, if we want to keep the number of iterations in one pass of VNDS approximately the same as in the standard ordering, i.e., if we want to use the same value for parameter d, then the subproblems examined will be larger than with the standard ordering, since the value of q will be smaller (see line 8 in Algorithm 19). The pseudo-code of this variant of VNDS-PC, denoted as VNDS-PC2, is provided in Algorithm 19.

3.6.2.2 A Double Decomposition Scheme

In this section we propose the use of a second level decomposition scheme within VNDS for the 0-1 MILP. The 0-1 MILP is tackled by decomposing the problem into several subproblems, where the number of binary variables with value 1 is fixed at a given integer value. Fixing the number of variables with value 1 to a given value $h \in \mathbb{N} \cup \{0\}$ can be achieved by adding the constraint $x_1 + x_2 + \ldots + x_p = h$, or, equivalently, $e^{\mathrm{T}}x = h$, where e is the vector of ones. Solving the 0-1 MILP by tackling separately each of the subproblems P_h for $h \in N$ appears to be an interesting approach for the case of the multidimensional knapsack problem [101], especially because the additional constraint $e^{\mathrm{T}}x = h$ provides tighter upper bounds than the classical LP-relaxation.

Function VNDS-PC2(P, d, x^*)
Choose stopping criteria (set *proceed*1=*proceed*2=`true`)
Add objective cut: $LB = c^{\mathrm{T}}x^*$; $P = (P \mid c^{\mathrm{T}}x > LB)$
while *proceed*1 **do**
  Find an optimal solution $\bar{x}$ of LP(P); set $UB = c^{\mathrm{T}}\bar{x}$
  if $(B(\bar{x}) = \mathscr{B})$ **then**
    break
  Find optimal solutions $\bar{x}^-$ of $LP^-_{x^*}$ and $\bar{x}^+$ of $LP^+_{x^*}$
  $\delta_j = \mid \bar{x}^-_j - \bar{x}^+_j \mid$, $j = 1, ..., p$; index x_j so that $\delta_j \leq \delta_{j+1}$, $j = 1, ..., p-1$
  Set $q = \mid \{ j \in \mathscr{B} \mid \delta_j \neq 0 \} \mid$, $k_{step} = \lfloor q/d \rfloor$, $k = p - k_{step}$
  while *proceed*2 **and** $k \geq 0$ **do**
    $J_k = \{1, \ldots, k\}$; $x' =$ MIPSOLVE$(P(x^*, J_k), x^*)$;
    if $(c^{\mathrm{T}}x' > c^{\mathrm{T}}x^*)$ **then**
      Update objective cut: $LB = c^{\mathrm{T}}x'$; $P = (P \mid c^{\mathrm{T}}x > LB)$;
      $x^* =$ LocalSearch(P, x'); $LB = c^{\mathrm{T}}x^*$; **break**
    else
      if $(k - k_{step} > p - q)$ **then**
        $k_{step} = \max\{\lfloor k/2 \rfloor, 1\}$
      Set $k = k - k_{step}$
    Update *proceed*2
  $x' =$ MIPSOLVE$(P(\bar{x}, B(\bar{x})), x^*)$; $LB = \max\{LB, c^{\mathrm{T}}x'\}$;
  Add pseudo-cut to P : $P = (P \mid \delta(B(\bar{x}), x, \bar{x}) \geq 1)$;
  $x' =$ MIPSOLVE$(P(\bar{x}^-, B(\bar{x}^-)), x^*)$; $LB = \max\{LB, c^{\mathrm{T}}x'\}$;
  Add pseudo-cut to P : $P = (P \mid \delta(B(\bar{x}^-), x, \bar{x}^-) \geq 1)$;
  $x' =$ MIPSOLVE$(P(\bar{x}^+, B(\bar{x}^+)), x^*)$; $LB = \max\{LB, c^{\mathrm{T}}x'\}$;
  Add pseudo-cut to P : $P = (P \mid \delta(B(\bar{x}^+), x, \bar{x}^+) \geq 1)$;
  Update *proceed*1;
return LB, UB, x^*.

Algorithm 19: VNDS for MIPs with pseudo-cuts and another ordering strategy

Formally, let P_h be the subproblem obtained from the original problem by adding the hyperplane constraint $e^{\mathrm{T}}x = h$ for $h \in N$, and enriched by an objective cut:

$$(P_h) \left[\begin{array}{l} \max\ c^{\mathrm{T}}x \\ \text{s.t.: } Ax \leq b \\ \qquad c^{\mathrm{T}}x \geq LB + 1 \\ \qquad e^{\mathrm{T}}x = h \\ \qquad x \in \{0,1\}^p \times \mathbb{R}^{n-p}_+ \end{array} \right.$$

Let h_{min} and h_{max} denote lower and upper bounds on the number of variables with value 1 in an optimal solution of the problem. Then it is obvious that $v(P) = \max\{v(P_h) \mid h_{min} \leq h \leq h_{max}\}$. Bounds $h_{min} = \lceil v(LP_0^-) \rceil$ and $h_{max} = \lfloor v(LP_0^+) \rfloor$ can be computed by solving the following two problems:

$$(LP_0^-)\begin{bmatrix}\min e^{\mathrm{T}}x\\ \text{s.t.: } Ax \le b\\ \quad c^{\mathrm{T}}x \ge LB+1\\ \quad x \in [0,1]^p \times \mathbb{R}_+^{n-p}\end{bmatrix} \qquad (LP_0^+)\begin{bmatrix}\max e^{\mathrm{T}}x\\ \text{s.t.: } Ax \le b\\ \quad c^{\mathrm{T}}x \ge LB+1\\ \quad x \in [0,1]^p \times \mathbb{R}_+^{n-p}\end{bmatrix}$$

We define the order of the hyperplanes at the beginning of the algorithm, and then we explore them one by one, in that order. The ordering can be done according to the objective values of the linear programming relaxations LP(P_h), $h \in H = \{h_{min}, \ldots, h_{max}\}$. In each hyperplane, `VNDS-PC1` is applied and if there is no improvement, the next hyperplane is explored. We refer to this method as `VNDDS` (short for *Variable Neighborhood Double Decomposition Search*), which corresponds to the pseudo-code in Algorithm 20. This idea is inspired by the approach proposed in [91], where the ordering of the neighborhood structures in Variable Neighborhood Descent is determined dynamically, by solving relaxations of the problems. Problems differ in one constraint that defines the Hamming distance h ($h \in H = \{h_{min}, .., h_{max}\}$).

Function `VNDDS`(P, x^*, d)
1 Solve the LP-relaxation problems LP_0^- and LP_0^+;
  Set $h_{min} = \lceil v(LP_0^-) \rceil$ and $h_{max} = \lfloor v(LP_0^+) \rfloor$;
2 Sort the set of subproblems $\{P_{h_{min}}, \ldots, P_{h_{max}}\}$ so that
  $v(LP(P_h)) \le v(LP(P_{h+1}))$, $h_{min} \le h < h_{max}$;
3 Find initial integer feasible solution x^*;
4 **for** $(h = h_{min}; h \le h_{max}; h++)$ **do**
5 $x' =$ `VNDS-PC1`(P_h, d, x^*)
6 **if** $(c^{\mathrm{T}}x' > c^{\mathrm{T}}x^*)$ **then**
  $x^* = x'$
  return x^*.

Algorithm 20: Two levels of decomposition with hyperplanes ordering

It is important to note that the exact variant of `VNDDS`, i.e., without any limitations regarding the running time or the number of iterations, converges to an optimal solution in a finite number of steps [51].

3.6.2.3 Comparison

For comparison purposes, five algorithms are ranked according to their objective values for the MIP benchmark instances in MIPLIB [77] and the benchmark instances for the Maximum Knapsack Problem (MKP) in [21]. Tables 3.3 and 3.4 report the average differences between the ranks of every pair of algorithms for the MIPLIP and MKP test sets, respectively.

Table 3.3 Objective value average rank differences on the MIPLIB set

ALGORITHM (average rank)	CPLEX (2.14)	VNDS-MIP (1.95)	VNDS-PC1 (2.64)	VNDS-PC2 (3.64)	VNDDS (4.64)
CPLEX (2.14)	0.00	0.18	−0.50	−1.50	−2.50
VNDS-MIP (1.95)	−0.18	0.00	−0.68	−1.68	−2.68
VNDS-PC1 (2.64)	0.50	0.68	0.00	−1.00	−2.00
VNDS-PC2 (3.64)	1.50	1.68	1.00	0.00	−1.00
VNDDS (4.64)	2.50	2.68	2.00	1.00	0.00

Table 3.4 Objective value average rank differences on the MKP set

ALGORITHM (average rank)	CPLEX (2.86)	VNDS-MIP (3.09)	VNDS-PC1 (2.09)	VNDS-PC2 (3.23)	VNDDS (3.72)
CPLEX (2.86)	0.00	−0.23	0.77	−0.36	−0.86
VNDS-MIP (3.09)	0.23	0.00	1.00	−0.14	−0.64
VNDS-PC1 (2.09)	−0.77	−1.00	0.00	−1.14	−1.64
VNDS-PC2 (3.23)	0.36	0.14	1.14	0.00	−0.50
VNDDS (3.72)	0.86	0.64	1.64	0.50	0.00

It appears that `VNDS-MIP` outperforms the other four methods on MIPLIB instances, while for the MKP set, the best performance is obtained with the `VNDS-PC1` heuristic.

3.7 Variable Neighborhood Search for Continuous Global Optimization

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

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

where $x \in R^n$, $f : R^n \to R$, $g_i : R^n \to R$, $i = 1,2,\ldots,m$, and $h_i : R^n \to R$, $i = 1,2,\ldots,r$, are possibly nonlinear continuous functions, and $a, b \in R^n$ are the variable bounds. A box constraint GOP is defined when only the variable bound constraints are present in the model.

GOPs naturally arise 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, a search effort of global scope is needed to find the globally optimal solution.

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 can be applied. However, if X is not a convex set or f is not a convex function, we can have many local optima and the problem may not be solved with classical techniques. For solving (GOP), VNS has been used in two different ways: (1) with neighborhoods induced by using a ℓ_p norm; (2) without using a ℓ_p norm.

(i) **VNS with ℓ_p norm neighborhoods** [40, 75, 81, 84]. A natural approach in applying VNS for solving GOPs is to induce neighborhood structures $\mathcal{N}_k(x)$ from the ℓ_p metric given as:

$$\rho(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}, \quad p \in [1,\infty) \tag{3.3}$$

and

$$\rho(x,y) = \max_{1 \le i \le n} |x_i - y_i|, \quad p \to \infty. \tag{3.4}$$

The neighborhood $\mathcal{N}_k(x)$ denotes the set of solutions in the k-th neighborhood of x based on the metric ρ. It is defined as

$$\mathcal{N}_k(x) = \{y \in X \mid \rho(x,y) \le \rho_k\}, \tag{3.5}$$

or

$$\mathcal{N}_k(x) = \{y \in X \mid \rho_{k-1} < \rho(x,y) \le \rho_k\}, \tag{3.6}$$

where ρ_k, known as the radius of $\mathcal{N}_k(x)$, is monotonically increasing with k ($k \ge 2$).

For solving box constraint GOPs, both [40] and [75] use the neighborhoods as defined in (3.6). The basic differences between the two algorithms reported there are as follows: (1) in the procedure suggested in [75] the ℓ_∞ norm is used, while in [40] the choice of metric is either left to the analyst, or changed automatically in some predefined order; (2) the commercial solver SNOPT [47] is used as a local search procedure within VNS in [75], while in [40], the analyst may choose one out of six different convex minimizers. A VNS based heuristic for solving the generally constrained GOP is suggested in [84]. There, the problem is first transformed into a sequence of box constrained problems within the well known exterior point method:

$$\min_{a \le x \le b} F_{\mu,q}(x) = f(x) + \frac{1}{\mu} \sum_{i=1}^{m} (\max\{0, g_i(x)\})^q + \sum_{i=1}^{r} |h_i(x)|^q, \tag{3.7}$$

where μ and $q \ge 1$ are a positive penalty parameter and penalty exponent, respectively. Algorithm 21 outlines the steps for solving the box constraint subproblem as proposed in [84].

```
Function Glob-VNS (x*, k_max, t_max)
1  Select the set of neighborhood structures N_k, k = 1,...,k_max
2  Select the array of random distributions types and an initial point x* ∈ X
3  x ← x*, f* ← f(x), t ← 0
4  while t < t_max do
5      k ← 1
6      repeat
7          for all distribution types do
8              y ← Shake(x*, k) // Get y ∈ N_k(x*) at random
9              y' ← BestImprovment(y) // Apply LS to obtain a local minimum y'
10             if f(y') < f* then
11                 x* ← y', f* ← f(y'), go to line 5
12         k ← k + 1
       until k = k_max
13     t ← CpuTime()
```

Algorithm 21: VNS using a ℓ_p norm

The Glob-VNS procedure from Algorithm 21 contains the following parameters in addition to k_{max} and t_{max}: (1) *Values of radii* ρ_k, $k = 1,\ldots,k_{\max}$, which may be defined by the user or calculated automatically in the minimizing process; (2) *Geometry* of neighborhood structures $\mathcal{N}_k$, defined by the choice of metric. Usual choices are the ℓ_1, ℓ_2, and ℓ_∞ norms; (3) *Distribution* types used for obtaining random points y from $\mathcal{N}_k$ in the *Shaking* step. A uniform distribution in $\mathcal{N}_k$ is the obvious choice, but other distributions may lead to much better performance on some problems. Different choices of neighborhood structures and random point distributions lead to different VNS-based heuristics.

(ii) **VNS without using ℓ_p norm neighborhoods**. Two different neighborhoods, $N_1(x)$ and $N_2(x)$, are used in the VNS based heuristic suggested in [99]. In $N_1(x)$, r (a parameter) random directions from the current point x are generated and a one dimensional search along each direction is performed. The best point (out of r) is selected as a new starting solution for the next iteration, if it is better than the current one. If not, as in VND, the search is continued within the next neighborhood $N_2(x)$. The new point in $N_2(x)$ is obtained as follows. The current solution is moved for each x_j ($j = 1,\ldots,n$) by a value Δ_j, taken at random from the interval $(-\alpha,\alpha)$; i.e., $x_j^{(new)} = x_j + \Delta_j$ or $x_j^{(new)} = x_j - \Delta_j$. Points obtained by the plus or minus sign for each variable define the neighborhood $N_2(x)$. If a relative increase of 1% in the value of $x_j^{(new)}$ produces a better solution than $x^{(new)}$, the + sign is chosen; otherwise the $-$ sign is chosen.

Neighborhoods $N_1(x)$ and $N_2(x)$ are used for designing two algorithms. The first, called VND, iterates over these neighborhoods until there is no improvement in the solution value. In the second variant, a local search is performed with N_2 and k_{max} is set to 2 for the shaking step.

It is interesting to note that computational results reported by all VNS based heuristics were very promising. They usually outperformed other recent approaches from the literature.

3.8 Variable Neighborhood Programming (VNP): VNS for Automatic Programming

Building an intelligent machine is an old dream that, thanks to computers, begins to take shape. Automatic programming is an efficient technique that has led to important developments in the field of artificial intelligence. Genetic programming (GP) [73], inspired by the genetic algorithm (GA), is among the few evolutionary algorithms used to evolve a population of programs. The main difference between GP and GA is the representation of a solution. An individual in GA can be a string, while in GP, the individuals are programs. A tree is the usual way to represent a program in GP. For example, assume that the current solution of a problem is the following function:

$$f(x_1,\ldots,x_5) = \frac{x_1}{x_2+x_3} + x_4 - x_5.$$

Then the code (tree) that calculates f using GP may be represented as in Fig. 3.4a.

Elleuch et al. [41, 42] recently adapted VNS rules for solving automatic programming problems. They first suggested an extended solution representation by adding coefficients to variables. Each terminal node was attached to its own parameter value. These parameters give a weight for each terminal node, with values from the interval [0, 1]. This type of representation allows VNP to examine parameter values and the tree structure in the same iteration, increasing the probability for finding a good solution faster. Let $G = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ denote a parameter set. In Fig. 3.4b an example of a solution representation in VNP is illustrated.

(i) Neighborhood structures. Nine different neighborhood structures are proposed in [42] based on a tree representation. To save space, we will just mention some of them:

- $N_1(T)$—**Changing a node value operator.** This neighborhood preserves the tree structure and changes only the values of a functional or a terminal node. Each node has a set of allowed values from which one can be chosen. Let x_i be the current solution; then a neighbor x_{i+1} differs from x_i by just a single node. A move within this neighborhood is shown in Fig. 3.5.

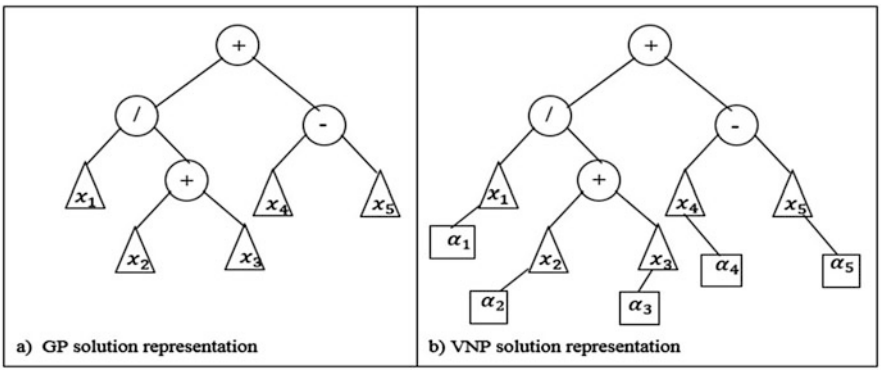

Fig. 3.4 Current solution representation in automatic programming problem: (**a**) $\frac{x_1}{x_2+x_3} + x_4 - x_5$; (**b**) $\frac{\alpha_1 x_1}{\alpha_2 x_2+\alpha_3 x_3} + \alpha_4 x_4 - \alpha_5 x_5$

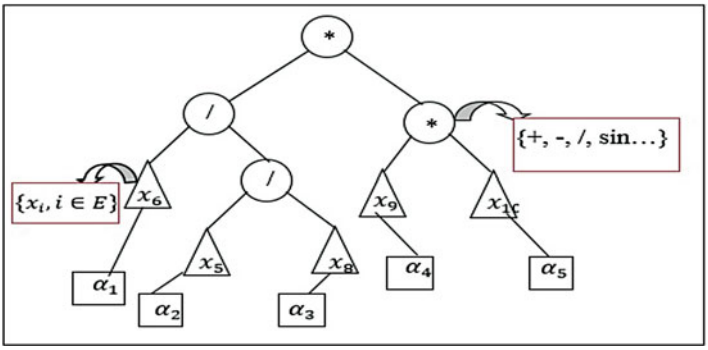

Fig. 3.5 Neighborhood N_1: changing a node value

- $N_2(T)$-**Swap operator**. Here, a subtree from the current tree is randomly selected and a new random subtree is generated as shown in Fig. 3.6a1 and a2. Then the new subtree replaces the current one (see Fig. 3.6b). In this move, any constraint related to the maximum tree size should be respected.

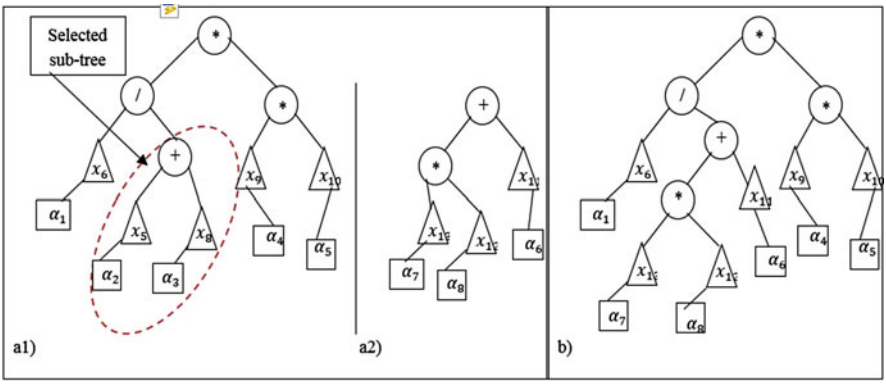

Fig. 3.6 Neighborhood N_2: swap operator. (**a1**) The current solution. (**a2**) New generated subtree. (**b**) The new solution

- $N_3(T)$—**Changing parameter values.** In the two previous neighborhoods, the tree structure and the node values were considered. In the $N_3(T)$ neighborhood, attention is paid to the parameters. So, the position and value of nodes are kept in order to search the neighbors in the parameter space. Figure 3.7 illustrates the procedure where the change from one value to another is performed at random.

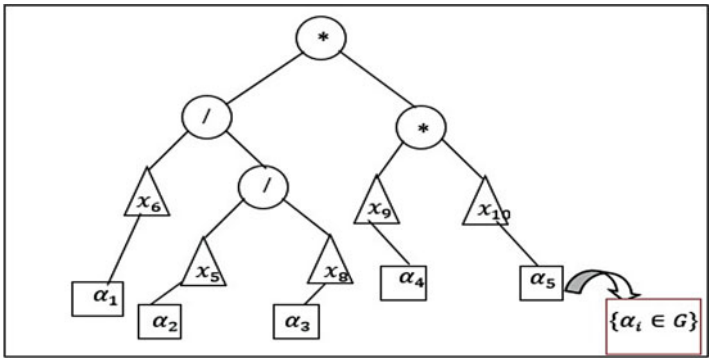

Fig. 3.7 Neighborhood N_3: change parameters

These neighborhoods may be used in both the local search step ($N_\ell, \ell \in [1, \ell_{max}]$) and in the shaking step ($\mathcal{N}_k, k \in [1, k_{max}]$) of the VNP.

(ii) VNP shaking. The shaking step allows diversification in the search space. The proposed VNP algorithm does not use exactly the same neighborhood structures N_ℓ than the local search. Thus, we denote the neighborhoods used in the shaking phase as $\mathcal{N}_k(T), k = 1, \ldots, k_{max}$. $\mathcal{N}_k(T)$ may be constructed by repeating k times one or more moves from the set $\{N_\ell(T), |\ell = 1, \ldots, \ell_{max}\}$. Consider, for example, the swap

operator $N_2(T)$. Let m denote the maximum number of nodes in the tree representation of the solution. We can get a solution from the kth neighborhood of T using the swap operator, where k represents the number of nodes of the new generated sub-tree. If n denotes the number of nodes in the original tree after deleting the old sub-tree, than $n+k \leq m$. The objective of the shaking phase is to provide a good starting point for the local search.

(iii) VNP objective function. The evaluation consists of defining a fitness (or objective) function to assess a solution. This function depends on the problem considered. After running each solution (program) on a training data set, the fitness may be measured by counting the training cases where the returned solution is correct or close to the exact solution.

(iv) An example: Time series forecasting (TSF) problem. Two widely used benchmark data sets of the TSF problem are considered in [42] to study the VNP capabilities: the Mackey-Glass series and the Box-Jenkins set. The parameters for the VNP implementation that were chosen after some preliminary testing are given in Table 3.5.

Table 3.5 VNP parameters adjustment for the forecasting problem

Parameters	Values
The functional set	$F = \{+, *, , pow\}$
The terminal sets	$\{(x_i, c), i \in [1, \ldots, m], m =$ number of inputs, $c \in R\}$
Neighborhood structures	$\{N_1, N_2, N_3\}$
Minimum tree length	20 nodes
Maximum tree length	200 nodes
Maximum number of iterations	50,000

The root mean square error (RMSE) is used as the fitness function, as it is normally done in the literature:

$$f(T) = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_t^j - y_{out}^j)^2}$$

where n is the total number of samples, and y_{out}^j and y_t^j are the output of the VNP model and the desired output for sample j, respectively. Next we illustrate with a comparison on a single Box-Jenkins instance.

The gas furnace data for this instance were collected from a combustion process of a methane air mixture [20]. This time series has found a widespread application as a benchmark example for testing prediction algorithms. The data set contains 296 pairs of input-output values. The input $u(t)$ corresponds to the gas flow, and the output $y(t)$ is the CO_2 concentration in the outlet gas. The inputs are $u(t-4)$, and $y(t-1)$, and the output is $y(t)$. In this work, 200 samples are used in the training phase and the remaining samples are used for the testing phase. The performance of the evolved VNP model is evaluated by comparing it with existing approaches.

The RMSE achieved by the VNP output model is (**0.00038**), which is better than the RMSE obtained by other approaches, as shown in Table 3.6.

Table 3.6 Comparison of testing error on Box-Jenkins dataset

Method	Prediction error RMSE
ODE [98]	0.5132
HHMDDE [38]	0.3745
FBBFNT [24]	0.0047
VNP [42]	**0.0038**

3.9 Discovery Science

In all the above applications, VNS is used as an optimization tool. It can also lead to results in "discovery science", i.e., for the development of new theories. This has been done for graph theory in a long series of papers with the common title "Variable neighborhood search for extremal graphs" that report on the development and applications of the AutoGraphiX (AGX) system [10, 28, 29]. This system addresses the following problems:

- Find a graph satisfying given constraints.
- Find optimal or near optimal graphs for an invariant subject to constraints.
- Refute a conjecture.
- Suggest a conjecture (or repair or sharpen one).
- Provide a proof (in simple cases) or suggest an idea of proof.

A basic idea then is to address all of these problems as parametric combinatorial optimization problems on the infinite set of all graphs (or in practice some smaller subset) using a generic heuristic to explore the solution space. This is being accomplished using VNS to find extremal graphs with a given number n of vertices (and possibly also a given number of edges). Extremal graphs may be viewed as a family of graphs that maximize some invariant such as the independence number or chromatic number, possibly subject to constraints. We may also be interested in finding lower and upper bounds on some invariant for a given family of graphs. Once an extremal graph is obtained, VND with many neighborhoods may be used to build other such graphs. Those neighborhoods are defined by modifications of the graphs such as the removal or addition of an edge, rotation of an edge, and so forth. Once a set of extremal graphs, parameterized by their order, is found, their properties are explored with various data mining techniques, leading to conjectures, refutations and simple proofs or ideas of proof.

More recent applications include [31, 32, 45, 50, 55] in chemistry, [8, 29] for finding conjectures, [16, 35] for largest eigenvalues, [23, 56, 64] for extremal values in graphs, independence [17, 18], specialty indexes [11, 15, 19, 61] and others [13, 60, 95, 96]. See [9] for a survey with many further references.

The current list of references in the series "VNS for extremal graphs" corresponds to [3, 8, 10–19, 23, 28, 29, 31, 32, 35, 45, 50, 55, 56, 60, 61, 64, 95, 96]. Another list of papers, not included in this series, is [4–7, 9, 30, 33, 49, 52–54, 65, 97]. Papers in these two lists cover a variety of topics:

1. **Principles of the approach** [28, 29] and its implementation [10];
2. **Applications to spectral graph theory**, e.g., bounds on the index for various families of graphs, graphs maximizing the index subject to some conditions [16, 19, 23, 35, 65];
3. **Studies of classical graph parameters**, e.g., independence, chromatic number, clique number, average distance [3, 9, 12, 17, 18, 95, 96];
4. **Studies of little known or new parameters of graphs**, e.g., irregularity, proximity and remoteness [4, 56];
5. **New families of graphs discovered by AGX**, e.g., bags, which are obtained from complete graphs by replacing an edge by a path, and bugs, which are obtained by cutting the paths of a bag [14, 60];
6. **Applications to mathematical chemistry**, e.g., study of chemical graph energy, and of the Randić index [11, 15, 32, 45, 49, 50, 52, 53, 55];
7. **Results of a systematic study of 20 graph invariants**, which led to almost 1500 new conjectures, more than half of which were proved by AGX and over 300 by various mathematicians [13];
8. **Refutation or strengthening of conjectures from the literature** [8, 30, 53];
9. **Surveys and discussions about various discovery systems in graph theory**, assessment of the state-of-the-art and the forms of interesting conjectures together with proposals for the design of more powerful systems [33, 54].

3.10 Conclusions

The general schemes of variable neighborhood search have been presented and discussed. In order to evaluate research development related to VNS, one needs a list of the desirable properties of metaheuristics.

1. *Simplicity:* the metaheuristic should be based on a simple and clear principle, which should be widely applicable;

2. *Precision:* the steps of the metaheuristic should be formulated in precise mathematical terms, independent of possible physical or biological analogies which may have been the initial source of inspiration;
3. *Coherence:* all steps of heuristics developed for solving a particular problem should follow naturally from the metaheuristic principles;
4. *Effectiveness:* heuristics for particular problems should provide optimal or near-optimal solutions for all known or at least the most realistic instances. Preferably, they should find optimal solutions for most benchmark problems for which such solutions are known;
5. *Efficiency:* heuristics for particular problems should take a moderate computing time to provide optimal or near-optimal solutions, or comparable or better solutions than the state-of-the-art;
6. *Robustness:* the performance of the metaheuristic should be consistent over a variety of instances, i.e., not merely fine-tuned to some training set and not so good elsewhere;
7. *User-friendliness:* the metaheuristic should be clearly expressed, easy to understand and, most importantly, easy to use. This implies it should have as few parameters as possible, ideally none;
8. *Innovation:* the principle of the metaheuristic and/or the efficiency and effectiveness of the heuristics derived from it should lead to new types of application.
9. *Generality:* the metaheuristic should lead to good results for a wide variety of problems;
10. *Interactivity:* the metaheuristic should allow the user to incorporate his knowledge to improve the resolution process;
11. *Multiplicity:* the metaheuristic should be able to produce several near optimal solutions from which the user can choose.

We have tried to show here that VNS possesses to a great extent, all of the above properties. This framework has led to heuristics which are among the very best ones for many problems. Interest in VNS is growing quickly. This is evidenced by the increasing number of papers published each year on this topic. 20 years ago, only a few; 15 years ago, about a dozen; 10 years ago, about 50, and more than 250 papers in 2016.

Figure 3.8 shows the parallel increase of the number of papers on VNS and on the other best known metaheuristics. Data are obtained by using the *Scopus search tool*, looking for the terms "Variable Neighborhood Search" (VNS) and "Metaheuristics" (MH). Figure 3.8 shows the number of times the terms appeared in the abstract of papers in this database. The years used are from 2000 to 2017 but in 2017 only the first 6 months (from January to June) are included. For comparison purposes, the number of papers with MH is divided by 4.

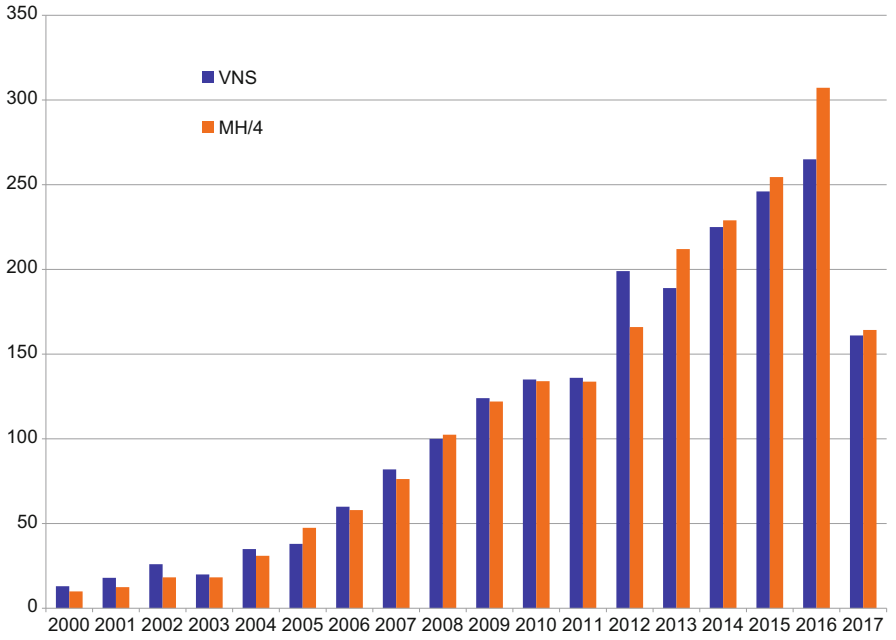

Fig. 3.8 VNS versus MH

Figure 3.9 shows the parallel increase of number of papers on VNS and on other most known Metaheuristics. Data are collected again from the *Scopus search tool* to look for the terms Variable Neighborhood Search (VNS), Tabu Search (TS), Genetic Algorithms (GA) and Simulated Annealing (SA). For better illustration, the number of appearances of TS, GA and SA are divided by 3, 50 and 10, respectively.

From the last figure, one can easily see that the relative increase in the number of papers with VNS is larger than the one of other major metaheuristics, especially in the last 5 years.

In addition, the 18th EURO Mini conference held in Tenerife in November 2005 was entirely devoted to VNS. It led to special issues of the *IMA Journal of Management Mathematics* in 2007 [76], *European Journal of Operational Research* [68] and *Journal of Heuristics* [87] in 2008. After that, VNS conferences took place in Herceg Novi—Montenegro (2012), Djerba—Tunis (2014), Málaga—Spain (2016) and in Ouro Preto—Brazil (2017). Each meeting was covered with before-conference Proceedings (in Electronic notes of Discrete Mathematics) and with at least one post-conference special issue in leading OR journals: Computers and OR, Journal of Global Optimization, IMA JMM, International Transactions of OR.

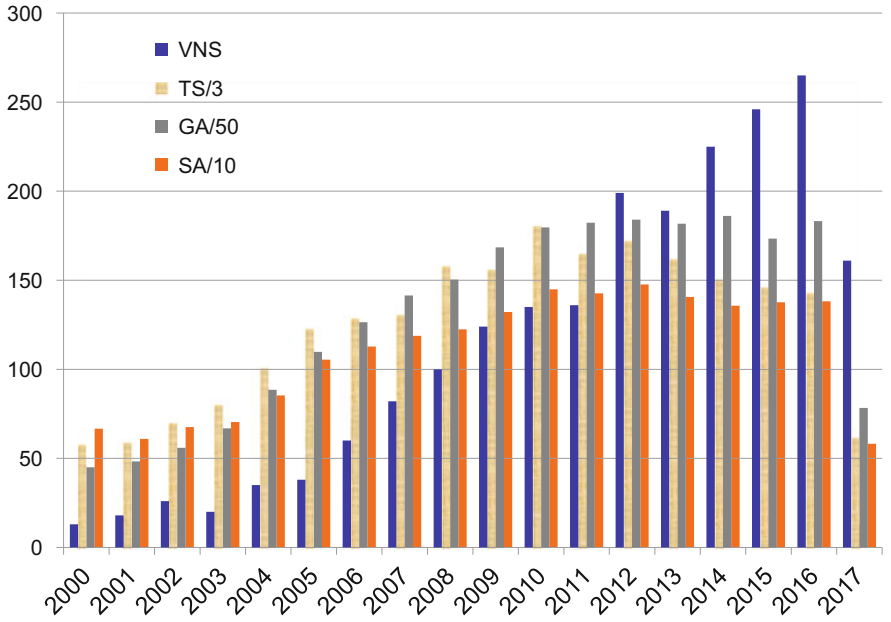

Fig. 3.9 VNS versus other main MHs

Acknowledgements The work of Nenad Mladenović was conducted at the National Research University Higher School of Economics, Nizhni Novgorod, Russia, and supported by RSF grant 14-41-00039. The fourth author is partially funded by Ministerio de Economía y Competitividad (Spanish Government) with FEDER funds, grant TIN2015-70226-R, and by Fundación Cajacanarias, grant 2016TUR19.

References

1. D.J. Aloise, D. Aloise, C.T.M. Rocha, C.C. Ribeiro, J.C. Ribeiro, L.S.S. Moura, Scheduling work-over rigs for onshore oil production. Discrete Appl. Math. **154**, 695–702 (2006)
2. D.V. Andrade, M.G.C. Resende, GRASP with path-relinking for network migration scheduling, in *Proceedings of International Network Optimization Conference (INOC)* (2007)
3. M. Aouchiche, P. Hansen, Recherche à voisinage variable de graphes extrèmes 13. À propos de la maille (French). RAIRO Oper. Res. **39**, 275–293 (2005)
4. M. Aouchiche, P. Hansen, Automated results and conjectures on average distance in graphs, in *Graph Theory in Paris*, ed. by A. Bondy, J. Fonlupt, J.L. Fouquet, J.C. Fournier, J.L. Ramírez Alfonsń. Trends in Mathematics (Birkhäuser, Basel, 2006), pp. 21–36
5. M. Aouchiche, P. Hansen, On a conjecture about the Randic index. Discrete Math. **307**, 262–265 (2007)
6. M. Aouchiche, P. Hansen, Bounding average distance using minimum degree. Graph Theory Notes N. Y. **56**, 21–29 (2009)
7. M. Aouchiche, P. Hansen, Nordhaus-Gaddum relations for proximity and remoteness in graphs. Comput. Math. Appl. **59**, 2827–2835 (2010)

8. M. Aouchiche, G. Caporossi, D. Cvetković, Variable neighborhood search for extremal graphs 8. Variations on Graffiti 105. Congressus Numerantium **148**, 129–144 (2001)
9. M. Aouchiche, G. Caporossi, P. Hansen, M. Laffay, AutoGraphiX: a survey. Electron Notes Discrete Math. **22**, 515–520 (2005)
10. M. Aouchiche, J.M. Bonnefoy, A. Fidahoussen, G. Caporossi, P. Hansen, L. Hiesse, J. Lacheré, A. Monhait, Variable neighborhood search for extremal graphs 14. The AutoGraphiX 2 system, in *Global Optimization: From Theory to Implementation*, ed. by L. Liberti, N. Maculan (Springer, Berlin, 2005), pp. 281–309
11. M. Aouchiche, P. Hansen, M. Zheng, Variable neighborhood search for extremal graphs 18. Conjectures and results about the Randic index. MATCH. Commun. Math. Comput. Chem. **56**, 541–550 (2006)
12. M. Aouchiche, O. Favaron, P. Hansen, Recherche à voisinage variable de graphes extrèmes 26. Nouveaux résultats sur la maille (French). Les Cahiers du GERAD, G-2007-55, 2007
13. M. Aouchiche, G. Caporossi, P. Hansen, Variable Neighborhood search for extremal graphs 20. Automated comparison of graph invariants. MATCH. Commun. Math. Comput. Chem. **58**, 365–384 (2007)
14. M. Aouchiche, G. Caporossi, P. Hansen, Variable neighborhood search for extremal graphs 27. Families of extremal graphs. Les Cahiers du GERAD, G-2007-87, 2007
15. M. Aouchiche, P. Hansen, M. Zheng, Variable neighborhood search for extremal graphs 19. Further conjectures and results about the Randic index. MATCH. Commun. Math. Comput. Chem. **58**, 83–102 (2007)
16. M. Aouchiche, F.K. Bell, D. Cvetković, P. Hansen, P. Rowlinson, S.K. Simić, D. Stevanović, Variable neighborhood search for extremal graphs 16. Some conjectures related to the largest eigenvalue of a graph. Eur. J. Oper. Res. **191**, 661–676 (2008)
17. M. Aouchiche, G. Brinkmann, P. Hansen, Variable neighborhood search for extremal graphs 21. Conjectures and results about the independence number. Discrete Appl. Math. **156**, 2530–2542 (2009)
18. M. Aouchiche, O. Favaron, P. Hansen, Variable neighborhood search for extremal graphs 22. Extending bounds for independence to upper irredundance. Discrete Appl. Math. **157**, 3497–3510 (2009)
19. M. Aouchiche, P. Hansen, D. Stevanović, Variable neighborhood search for extremal graphs 17. Further conjectures and results about the index. Discussiones Mathematicae: Graph Theory **29**, 15–37 (2009)
20. C. Audet, V. Bachard, S. Le Digabel, Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search. J. Glob. Optim. **41**, 299–318 (2008)
21. J. Beasley, OR-Library: distributing test problems by electronic mail. J. Oper. Res. Soc. **41**(11), 1069–1072 (1990)
22. N. Belacel, P. Hansen, N. Mladenović, Fuzzy J-means: a new heuristic for fuzzy clustering. Pattern Recognit. **35**, 2193–2200 (2002)
23. S. Belhaiza, de, N. Abreu, HanP. sen, C. Oliveira, Variable neighborhood search for extremal graphs 11. Bounds on algebraic connectivity, in *Graph Theory and Combinatorial Optimization*, ed. by D. Avis, A. Hertz, O. Marcotte (2007), pp. 1–16
24. S. Bouaziz, H. Dhahri, A.M. Alimi, A. Abraham, A hybrid learning algorithm for evolving flexible beta basis function neural tree model. Neurocomputing **117**, 107–117 (2013)
25. J. Brimberg, N. Mladenović, A variable neighborhood algorithm for solving the continuous location-allocation problem. Stud. Locat. Anal. **10**, 1–12 (1996)
26. J. Brimberg, P. Hansen, N. Mladenović, É. Taillard, Improvements and comparison of heuristics for solving the multisource Weber problem. Oper. Res. **48**, 444–460 (2000)
27. S. Canuto, M. Resende, C. Ribeiro, Local search with perturbations for the prize-collecting Steiner tree problem in graphs. Networks **31**, 201–206 (2001)
28. G. Caporossi, P. Hansen, Variable neighborhood search for extremal graphs 1. The AutoGraphiX system. Discrete Math. **212**, 29–44 (2000)
29. G. Caporossi, P. Hansen, Variable neighborhood search for extremal graphs 5. Three ways to automate finding conjectures. Discrete Math. **276**, 81–94 (2004)

30. G. Caporossi, A.A. Dobrynin, I. Gutman, P. Hansen, Trees with palindromic Hosoya polynomials. Graph Theory Notes N. Y. **37**, 10–16 (1999)
31. G. Caporossi, D. Cvetković, I. Gutman, P. Hansen, Variable neighborhood search for extremal graphs 2. Finding graphs with extremal energy. J. Chem. Inform. Comput. Sci. **39**, 984–996 (1999)
32. G. Caporossi, I. Gutman, P. Hansen, Variable neighborhood search for extremal graphs 4. Chemical trees with extremal connectivity index. Comput. Chem. **23**, 469–477 (1999)
33. G. Caporossi, I. Gutman, P. Hansen, L. Pavlović, Graphs with maximum connectivity index. Comput. Biol. Chem. **27**, 85–90 (2003)
34. J. Cohoon, S. Sahni, Heuristics for backplane ordering. J. VLSI Comput. Syst. **2**, 37–61 (1987)
35. D. Cvetkovic, S. Simic, G. Caporossi, P. Hansen, Variable neighborhood search for extremal graphs 3. On the largest eigenvalue of color-constrained trees. Linear Multilinear Algebra **49**, 143–160 (2001)
36. W.C. Davidon, Variable metric algorithm for minimization. Argonne National Laboratory Report ANL-5990 (1959)
37. J. Desrosiers, N. Mladenović, D. Villeneuve, Design of balanced MBA student teams. J. Oper. Res. Soc. **56**, 60–66 (2005)
38. H. Dhahri, A.M. Alimi, A. Abraham, Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network. Neurocomputing **97**, 131–140 (2012)
39. A. Djenic, N. Radojicic, M. Maric, N. Mladenović, Parallel VNS for bus terminal location problem. Appl. Soft Comput. **42**, 448–458 (2016)
40. M. Dražić, V. Kovacevic-Vujcić, M. Cangalović, N. Mladenović, GLOB - a new VNS-based software for global optimization, in *Global Optimization: From Theory to Implementation*, ed. by L. Liberti, N. Maculan (Springer, Berlin, 2006), pp. 135–144
41. S. Elleuch, B. Jarboui, N. Mladenović, Reduced variable neighborhood programming for the preventive maintenance planning of railway infrastructure. GERAD Technical report, G-2016-92, Montreal (2016)
42. S. Elleuch, B. Jarboui, N. Mladenović, Variable neighborhood programming - A new automatic programming method in artificial intelligence. GERAD Technical report, G-2016-21, Montreal (2016)
43. M. Fischetti, A. Lodi, Local branching. Math. Program. **98**, 23–47 (2003)
44. R. Fletcher, M.J.D. Powell, Rapidly convergent descent method for minimization. Comput. J. **6**, 163–168 (1963)
45. P.W. Fowler, P. Hansen, G. Caporossi, A. Soncini, Variable neighborhood search for extremal graphs 7. Polyenes with maximum HOMO-LUMO gap. Chem. Phys. Lett. **49**, 143–146 (2001)
46. M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, New York, 1978)
47. P. Gill, W. Murray, M.A. Saunders, SNOPT: an SQP algorithms for largescale constrained optimization. SIAM J. Optim. **12**, 979–1006 (2002)
48. R.E. Griffith, R.A. Stewart, A nonlinear programming technique for the optimization of continuous processing systems. Manag. Sci. **7**, 379–392 (1961)
49. I. Gutman, O. Miljković, G. Caporossi, P. Hansen, Alkanes with small and large Randić connectivity indices. Chem. Phys. Lett. **306**, 366–372 (1999)
50. I. Gutman, P. Hansen, H. Mélot, Variable neighborhood search for extremal graphs 10. Comparison of irregularity indices for chemical trees. J. Chem. Inform. Model. **45**, 222–230 (2005)
51. S. Hanafi, J. Lazić, N. Mladenović, C. Wilbaut, I. Crévits, New variable neighborhood search based 0-1 MIP heuristic. Yugoslav J. Oper. Res. **25**, 343–360 (2015)
52. P. Hansen, Computers in graph theory. Graph Theory Notes N. Y. **XLIII**, 20–39 (2002)
53. P. Hansen, How far is, should and could be conjecture-making in graph theory an automated process? in *Graph and Discovery*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol.69 (American Mathematical Society, Providence, 2005), pp. 189–229
54. P. Hansen, H. Mélot, Computers and discovery in algebraic graph theory. Linear Algebra Appl. **356**, 211–230 (2002)

55. P. Hansen, H. Mélot, Variable neighborhood search for extremal graphs 6. Analyzing bounds for the connectivity index. J. Chem. Inform. Comput. Sci. **43**, 1–14 (2003)
56. P. Hansen, H. Mélot, Variable neighborhood search for extremal graphs 9. Bounding the irregularity of a graph. Graphs Discov. **69**, 253–264 (2005)
57. P. Hansen, N. Mladenović, J-Means: a new local search heuristic for minimum sum-of-squares clustering. Pattern Recognit. **34**, 405–413 (2001)
58. P. Hansen, N. Mladenović, Variable neighborhood search: principles and applications. Eur. J. Oper. Res. **130**, 449–467 (2001)
59. P. Hansen, N. Mladenović, Variable neighborhood search, in *Handbook of Metaheuristics*, ed. by F. Glover, G. Kochenberger (Kluwer, Boston, 2003), pp. 145–184
60. P. Hansen, D. Stevanović, Variable neighborhood search for extremal graphs 15. On bags and bugs. Discrete Appl. Math. **156**, 986–997 (2005)
61. P. Hansen, D. Vukičević, Variable neighborhood search for extremal graphs 23. On the Randic index and the chromatic number. Discrete Math. **309**, 4228–4234 (2009)
62. P. Hansen, B. Jaumard, N. Mladenović, A. Parreira, Variable neighborhood search for weighted maximum satisfiability problem. Les Cahiers du GERAD, G-2000-62, 2000
63. P. Hansen, N. Mladenović, D. Pérez-Brito, Variable neighborhood decomposition search. J. Heuristics **7**, 335–350 (2001)
64. P. Hansen, H. Mélot, I. Gutman, Variable neighborhood search for extremal graphs 12. A note on the variance of bounded degrees in graphs. MATCH Commun. Math. Comput. Chem. **54**, 221–232 (2005)
65. P. Hansen, M. Aouchiche, G. Caporossi, H. Mélot, D. Stevanović, What forms do interesting conjectures have in graph theory? in *Graph and Discovery*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 69 (American Mathematical Society, Providence, 2005), pp. 231–251
66. P. Hansen, N. Mladenović, D. Urošević, Variable neighborhood search and local branching. Comput. Oper. Res. **33**, 3034–3045 (2006)
67. P. Hansen, J. Brimberg, D. Urošević, N. Mladenović, Primal-dual variable neighborhood search for the simple plant location problem. INFORMS J. Comput. **19**, 552–564 (2007)
68. P. Hansen, N. Mladenović, J.A. Moreno Pérez, Variable neighborhood search. Eur. J. Oper. Res. **191**, 593–595 (2008)
69. P. Hansen, N. Mladenović, J.A. Moreno Pérez, Variable neighborhood search: methods and applications. 4OR. Q. J. Oper. Res. **6**, 319–360 (2008)
70. A. Hertz, M. Plumettaz, N. Zufferey, Variable space search for graph coloring. Discrete Appl. Math. **156**, 2551–2560 (2008)
71. ILOG CPLEX 10.1. User's Manual (2006)
72. K. Jornsten, A. Lokketangen, Tabu search for weighted k-cardinality trees. Asia-Pacific J. Oper. Res. **14**, 9–26 (1997)
73. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT, Cambridge, 1992)
74. J. Lazić, S. Hanafi, N. Mladenović, D. Urošević, Variable neighbourhood decomposition search for 0–1 mixed integer programs. Comput. Oper. Res. **37**, 1055–1067 (2010)
75. L. Liberti, M. Dražić, Variable neighbourhood search for the global optimization of constrained NLPs, in *Proceedings of GO Workshop*, Almeria, 2005
76. B. Melián, N. Mladenović, Editorial. IMA J. Manag. Math. **18**, 99–100 (2007)
77. MIPLIB http://miplib.zib.de/miplib2003/
78. N. Mladenović, A variable neighborhood algorithm – a new metaheuristic for combinatorial optimization. Abstracts of papers presented at Optimization Days, Montréal (1995), p. 112
79. N. Mladenović, Formulation space search – a new approach to optimization (plenary talk), in *Proceedings of XXXII SYMOPIS'05*, ed. by J. Vuleta (Vrnjacka Banja, Serbia, 2005)
80. N. Mladenović, P. Hansen, Variable neighborhood search. Comput. Oper. Res. **24**, 1097–1100 (1997)
81. N. Mladenović, J. Petrović, V. Kovačević-Vujčić, M. Čangalović, Solving spread spectrum radar polyphase code design problem by tabu search and variable neighborhood search. Eur. J. Oper. Res. **151**, 389–399 (2003)

82. N. Mladenović, F. Plastria, D. Urošević, Reformulation descent applied to circle packing problems. Comput. Oper. Res. **32**, 2419–2434 (2005)
83. N. Mladenović, F. Plastria, D. Urošević, Formulation space search for circle packing problems. Lect. Notes Comput. Sci. **4638**, 212–216 (2007)
84. N. Mladenović, M. Dražić, V. Kovačevic-Vujčić, M. Čangalović, General variable neighborhood search for the continuous optimization. Eur. J. Oper. Res. **191**, 753–770 (2008)
85. N. Mladenović, D. Urošević, D. Pérez-Brito, C.G. García-González, Variable neighbourhood search for bandwidth reduction. Eur. J. Oper. Res. **200**, 14–27 (2010)
86. N. Mladenovic, J. Kratica, V. Kovacevic-Vujcic, M. Cangalovic, Variable neighborhood search for metric dimension and minimal doubly resolving set problems. Eur. J. Oper. Res. **220**, 328–337 (2012)
87. J.M. Moreno-Vega, B. Melián, Introduction to the special issue on variable neighborhood search. J. Heuristics **14**, 403–404 (2008)
88. J.J. Pantrigo, R. Marti, A. Duarte, E.G. Pardo, Scatter search for the cutwidth minimization problem. Ann. Oper. Res. **199**, 285–304 (2012)
89. E.G. Pardo, N. Mladenović, J.J. Pantrigo, A. Duarte, Variable formulation search for the cutwidth minimization problem. Appl. Soft Comput. **13**, 2242–2252 (2014)
90. F. Plastria, N. Mladenović, D. Urošević, Variable neighborhood formulation space search for circle packing, in *18th Mini Euro Conference VNS*, Tenerife, 2005
91. J. Puchinger, G. Raidl, Bringing order into the neighborhoods: relaxation guided variable neighborhood search. J. Heuristics **14**, 457–472 (2008)
92. C.C. Ribeiro, M.C. de Souza, Variable neighborhood search for the degree-constrained minimum spanning tree problem. Discrete Appl. Math. **118**, 43–54 (2002)
93. C.C. Ribeiro, E. Uchoa, R. Werneck, A hybrid GRASP with perturbations for the Steiner problem in graphs. INFORMS J. Comput. **14**, 228–246 (2002)
94. J. Rolim, O. Sýkora, I. Vrt'o, Optimal cutwidths and bisection widths of 2- and 3-dimensional meshes, in *Graph-Theoretic Concepts in Computer Science*. Lecture Notes in Computer Science, vol. 1017 (1995), pp. 252–264
95. J. Sedlar, D. Vukicevic, M. Aouchiche, P. Hansen, Variable neighborhood search for extremal graphs 24. Conjectures and results about the clique number. Les Cahiers du GERAD G-2007-33, 2007
96. J. Sedlar, D. Vukicevic, M. Aouchiche, P. Hansen, Variable neighborhood search for extremal graphs 25. Products of connectivity and distance measures. Les Cahiers du GERAD, G-2007-47, 2007
97. D. Stevanovic, M. Aouchiche, P. Hansen, On the spectral radius of graphs with a given domination number. Linear Algebra Appl. **428**, 1854–1864 (2008)
98. B. Subudhi, D. Jena, A differential evolution based neural network approach to nonlinear system identification. Appl. Soft Comput. **11**, 861–871 (2011)
99. A.D. Toksari, E. Güner, Solving the unconstrained optimization problem by a variable neighborhood search. J. Math. Anal. Appl. **328**, 1178–1187 (2007)
100. D. Urošević, J. Brimberg, N. Mladenović, Variable neighborhood decomposition search for the edge weighted *k*-cardinality tree problem. Comput. Oper. Res. **31**, 1205–1213 (2004)
101. Y. Vimont, S. Boussier, M. Vasquez, Reduced costs propagation in an efficient implicit enumeration for the 01 multidimensional knapsack problem. J. Comb. Optim. **15**, 165–178 (2008)
102. R. Whitaker, A fast algorithm for the greedy interchange of large-scale clustering and median location problems. INFOR **21**, 95–108 (1983)