Université de Valenciennes et du Hainaut-Cambrésis

Thèse de doctorat

Pour obtenir le grade de Docteur de l'Université de

VALENCIENNES ET DU HAINAUT-CAMBRESIS

Sciences et Technologie, Mention : Informatique :

Présentée et soutenue par Raca Todosijević.

Le 22 juin 2015, à Valenciennes

Ecole doctorale :

Sciences Pour l'Ingénieur (SPI)

Equipe de recherche, Laboratoire :

Laboratoire d'Automatique, de Mécanique et d'Informatique Industrielles et Humaines (LAMIH)

Theoretical and Practical Contributions on Scatter Search, Variable Neighborhood Search and Matheuristics for 0-1 Mixed Integer Programs

Contributions théoriques et pratiques pour la Recherche Dispersée, Recherche à Voisinage Variable et Matheuristque pour les programmes en nombres entiers mixtes

JURY

Président du jury

- Fréville, Arnaud. Professeur, Université de Valenciennes, France

Rapporteurs

- Hansen, Pierre. Professeur, HEC Montréal, Canada
- Punnen, Abraham. Professeur, Simon Fraser University, Canada

Examinateurs

- Glover, Fred. Professeur, University of Colorado, USA
- Mladenovic, Nenad. Professeur, Université de Valenciennes, France
- Vasquez, Michel. Professeur, Ecole des mînes d'Alès, France

Directeurs de thèse

- Hanafi, Saïd. Professeur, Université de Valenciennes, France
- Gendron, Bernard. Professeur, Université de Montréal, Canada

Abstract

This thesis consists of results obtained studying Scatter Search, Variable Neighbourhood Search (VNS), and Matheuristics in both theoretical and practical context. Regarding theoretical results, one of the main contribution of this thesis is a convergent scatter search with directional rounding algorithm for 0-1 Mixed Integer Programs (MIP) with the proof of its finite convergence. Besides this, a convergent scatter search algorithm is accompanied by two variants of its implementation. Additionally, several scatter search based heuristics, stemming from a convergent scatter search algorithm have been proposed and tested on some instances of 0-1 MIP. The versions of the methods tested are first stage implementations to establish the power of the methods in a simplified form. Our findings demonstrate the efficacy of these first stage methods, which makes them attractive for use in situations where very high quality solutions are sought with an efficient investment of computational effort.

This thesis also includes new variants of Variable Neighborhood Search metaheuristic such as a two-level variable neighborhood search, a nested variable neighborhood search, a cyclic variable neighborhood descent and a variable neighborhood diving. Additionally, several efficient implementation of those variable neighborhood search algorithms have been successfully applied for solving NP-Hard problems appearing in transportation, logistics, power generation, scheduling and clustering. On all tested problems, the proposed VNS heuristics turned out to be a new state-of-the art heuristics.

The last contribution of this thesis consists of proposing several matheuristics for solving Fixed-Charge Multicommodity Network Design (MCND) problem. The performances of these matheuristics have been disclosed on benchmark instances for MCND. The obtained results demonstrate the competitiveness of the proposed matheuristics with other existing approaches in the literature.

Keywords:Optimization – Scatter Search – Variable Neighborhood Search – Heuristic – Metaheuristic – Matheuristic – Directional rounding – Convergence – 0-1 MIP – Routing – TSP – VRP – Location – Clustering – Scheduling – Maintenance – Unit commitment – Network Design.

Résumé

Cette thèse comporte des résultats théoriques et pratiques sur deux métaheuristiques, la Recherche Dispersée et la Recherche Voisinage variable (RVV), ainsi que sur des Matheuristiques. Au niveau théorique, la contribution principale de cette thèse est la proposition dun algorithme de recherche dispersée avec larrondi directionnel convergent pour les programmes en nombres entiers mixtes (0-1 MIP), avec une preuve de cette convergence en un nombre fini ditérations. En se basant sur cet algorithme convergeant, deux implémentations et plusieurs heuristiques sont proposées et testées sur des instances de 0-1 MIP. Les versions testées reposent sur des implémentations non optimisées pour mettre en évidence la puissance des approches dans une forme simplifiée. Nos résultats démontrent lefficacité de ces approches initiales, ce qui les rend attractives lorsque des solutions de très haute qualité sont recherchées avec un investissement approprié en termes deffort de calcul. Cette thèse inclut également quelques nouvelles variantes de la métaheuristique Recherche Voisinage Variable telles quune recherche voisinage variable deux niveaux, une recherche voisinage variable imbriquée, une descente voisinage variable cyclique et une heuristique de plongée voisinage variable. En outre, plusieurs implémentations efficaces de ces algorithmes basés sur la recherche voisinage variable ont été appliquées avec succès des problèmes NP-Difficiles apparaissant en transport, logistique, production dénergie, ordonnancement, et segmentation. Les heuristiques proposées se sont avérées tre les nouvelles heuristiques de référence sur tous les problèmes considérés. La dernière contribution de cette thèse repose sur la proposition de plusieurs matheuristiques pour résoudre le problème de Conception de Réseau Multi-flots avec Cot fixe (CRMC). Les performances de ces matheuristiques ont été évaluées sur un ensemble dinstances de référence du CRMC. Les résultats obtenus démontrent la compétitivité des approches proposées par rapport aux approches existantes de la littérature.

Mots Clés: Optimisation - Recherche Dispersée - Recherche Voisinage Variable - Heuristique - Métaheuristique - Matheuristique - Arrondi Directionnel - Convergence - 0-1 MIP - Routage - Voyageur de Commerce - Elaboration de tournées - Localisation - Clustering - Ordonnancement - Maintenance - Affectation des Unités de production dénergie - Conception de réseau.

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

The 0-1 mixed integer programming (MIP) problem is used for modeling many combinatorial problems, ranging from logical design to scheduling and routing as well as encompassing graph theory models for resource allocation and financial planning. Many 0-1 MIP problems are NP hard, so exact methods (e.g., Branch and Bound, Branch and Cut, Branch and price and so on) are not suitable for solving large scale instances. Namely, the obtention of optimal solutions for majority of NP hard optimization problems is not possible in a realistic or justifiable amount of resources consumption (time, memory and so on). More precisely, often exact methods succeed to find near-optimal solution quickly but need a lot of time to reach the optimal one. Additionally, even if they succeed to reach an optimal solution quickly they sometimes consume a significant portion of the total solution time to prove its optimality. For these reasons, in many practical settings, exact methods are used as heuristics, stopping them before getting a proof of optimality (e.g., imposing CPU time limit, maximum number of iterations to be performed, node limit). These drawbacks of exact methods have attracted researchers to develop many heuristic methods to tackle NP hard optimization problems. The advantage of heuristics is not only their ability to produce high-quality solutions in short time, but also the fact that they can be easily combined with exact methods to speed them up. The early incumbent solutions produced by a heuristic can help a Branch and Bound algorithm to reduce the amount of memory needed to store the Branch-and-Bound tree and accelerate the exploration of the Branch-and-Bound tree.

In this thesis, scatter search, variable neighborhood search and matheuristics for solving 0-1 Mixed Integer Programs are studied in both theoretical and practical context. Scatter search, variable neighborhood search and matheuristics represent a flexible frameworks for building heuristics for approximately solving combinatorial and non-linear continuous optimization problems. Scatter Search (SS) is an evolutionary metaheuristic introduced by Fred Glover (1977) as a heuristic for integer programming. It combines decision rules and problem constraints, and it has its origins in surrogate constraint strategies.

Scatter Search, unlike Genetic Algorithms, operates on a small set of solutions and makes only limited use of randomization as a proxy for diversification when searching for a globally optimal solution. Variable neighborhood search metaheuristic was proposed by Nenad Mladenovic and Pierre Hansen in 1997. It is based on systematic changing of neighborhood structures during the search for an optimal (or near-optimal) solution. Additionally, matheuristics that combines exact and heuristic approaches are proposed for 0-1 MIP and the Fixed-Charge Multicommodity Network Design. This thesis is structured in four main chapters.

Chapter 1 contains the review of heuristics for 0-1 MIP problems. The survey is focused on general heuristics for 0-1 MIP problems, not those that exploit the structure of the considered problem (i.e., Lagrangian based heuristics, heuristics proposed for various special combinatorial structures, like scheduling and location problems, the traveling salesman problem, knapsack problems, etc.). Additionally, we do not provide the review of general heuristics for solving classes of problems that include 0-1 MIP problems as a special case such as heuristics for general MIP, convex integer programming and so on. More precisely, we are strictly focused on the stand-alone heuristics for 0-1 MIP as well as those heuristics that use linear programming techniques or solve series of linear programming models or reduced problems, deduced from the initial one, in order to produce a high quality solution of the considered problem. The emphasis is on how mathematical programming techniques can be used for approximate problem solving, rather than on a comparing performances of heuristics. For most of considered heuristics we not only describe their main ideas and their work but also provide their pseudo-codes written in the same framework. Most of these heuristics are already embedded in many commercial solvers such as CPLEX, Gurobi, GAMS, XPRESS etc., but there is still a need for new heuristics that will additionally accelerate exact methods.

The contributions on Variable Neighborhood Search (VNS) metaheuristic are presented in Chapter 2. The chapter starts by describing the framework of a VNS and existing and newly proposed VNS variants such as a two-level VNS, a nested VNS, and a cyclic variable neighborhood descent. The rest of this chapter is dedicated to efficient implementation of VNS heuristics

that we developed for solving optimization problems such as the traveling salesman problem with draft Limits, the traveling salesman problem with times windows, the attractive traveling salesman problem, the swap-body vehicle routing problem, the unit commitment problem, the periodic maintenance problem and the minimum sum of squares clustering on networks. On all tested problems, the proposed VNS heuristics turned out to be new state-of-the art heuristics.

Chapter 3 describes several matheurstic approaches. First, two new diving heuristics are proposed for finding a feasible solution for a MIP problem, called Variable neighborhood (VN) diving and Single neighborhood (SN) diving, respectively. They perform systematic hard variable fixing (i.e., diving) by exploiting the information obtained from a series of LP relaxations in order to generate a sequence of subproblems. Pseudo cuts are added during the search process to avoid revisiting the same search space areas. VN diving is based on the variable neighborhood decomposition search framework. Conversely, SN diving explores only a single neighborhood in each iteration: if a feasible solution is not found, then the next reference solution is chosen using the feasibility pump principle and the search history. Moreover, we prove that the two proposed algorithms converge in a finite number of iterations. We show that our proposed algorithms significantly outperform the CPLEX 12.4 MIP solver and the recent variants of feasibility pump regarding the solution quality. On the other hand, we propose several iterative linear programming-based heuristics for solving Fixed-Charge Multicommodity Network Design (MCND) problem. We propose how to adapt well-known Slope Scaling heuristic for MCND in order to tackle reduced problems of MCND obtained by fixing some binary variables. Moreover, we show that ideas of a convergent algorithm based on the LP-relaxation and pseudo-cuts may be used to guide Slope Scaling heuristic during the search for an optimal (or near-optimal) solution and vice-versa. The results obtained testing proposed approaches on the benchmark instances disclose their effectiveness and efficiency when compared with existing approaches in the literature.

Finally, in Chapter 4, we propose several algorithms of Scatter Search with directional rounding for 0-1 MIP. First, we address directional rounding both independently and together with other algorithmic components, studying

its properties as a mapping from continuous to discrete (binary) space. We establish several useful properties of directional rounding and show that it provides an extension of classical rounding and complementing operators. Moreover, we observe that directional rounding of a line, as embodied in a Star Path, contains a finite number of distinct 0–1 points. This property, together with those of the solution space of 0-1 MIP, enables us to organize the search for an optimal solution of 0–1 MIP problems using Scatter Search in association with both cutting plane and extreme point solution approaches. This chapter provides a convergent scatter search algorithm for 0-1 MIP with the proof of its finite convergence, accompanied by two variants of its implementation and examples that illustrate the running of the approach. Additionally, we present several heuristic approaches for finding good solution based on scatter search with directional rounding. Additionally, we conduct an empirical study in order to find out the best way for choosing ingredients of a heuristic that combines Scatter search and directional rounding. Finally, we disclose the merit of our approaches by computational testing on a test bed of 0-1 MIP problems. The versions of the methods tested are first stage implementations to establish the power of the methods in a simplified form. Our findings demonstrate the efficacy of these first stage methods, which makes them attractive for use in situations where very high quality solutions are sought with an efficient investment of computational effort.
Most of the contributions of this thesis have been published or submitted for possible publishing in international journals [36, 152, 169, 170, 209, 210, 211, 212, 213].

The thesis is written so that each chapter can be read independently, and therefore there is some slight overlapping among chapters. In addition, some of the contributions are presented briefly in order to have a thesis with reasonable number of pages. This is especially true for the contributions presented in Chapter 2. Note that all contributions of the thesis, presented in the form of papers, may be founded on the webpage `http://www.univ-valenciennes.fr/LAMIH/membres/todosijevic_raca`.

Chapter 1

Survey

1.1 Introduction

The zero–one mixed integer programming problem is used for modeling many combinatorial problems, ranging from logical design to scheduling and routing as well as encompassing graph theory models for resource allocation and financial planning. A 0–1 mixed integer program (MIP) may be written in the following form:

$$(MIP)\begin{cases} maximize\ v = cx \\ s.t.\ Ax \leq b \\ 0 \leq x_j \leq U_j,\ j \in N = \{1, \ldots, n\} \\ x_j \in \{0, 1\},\ j \in \mathcal{I} \subseteq N \end{cases} \tag{1.1}$$

where A is a $m \times n$ constant matrix, b is a constant vector, the set N denotes the index set of variables, while the set $\mathcal{I}$ contains indices of binary variables. Each variable x_j has an upper bound denoted by U_j (which equals 1 if x_j is binary variable, while otherwise may be infinite). It is assumed that all continuous variables can be represented (either directly or by transformation) as slack variables, i.e., the associated columns of the (possibly transformed) matrix A constitute an identity matrix. Hence, if the values of binary variables are known, the continuous variables receive their values automatically. The set of all slack variables will be denoted by $\mathcal{S}$. The integer problem defined in this manner will be denoted simply by MIP while the relaxation of MIP obtained by excluding integrality constraints will be denoted by LP. A feasible solution of $MIP(LP)$ will be called $MIP(LP)$ feasible. An optimal solution of the LP problem will be denoted by $\overline{x}$. The set of all MIP feasible solutions will be denoted by X, i.e., $X = \{x \in \mathbb{R}^n : Ax \leq b;\ 0 \leq x_j \leq$

$U_j,\ j \in N = \{1, \ldots, n\};\ x_j \in \{0, 1\},\ j \in \mathcal{I} \subseteq N\}$. An optimal solution or the best found solution obtained in an attempt to solve the MIP problem will be denoted by x^*, while its objective value will be denoted by v^*.

Since 0-1 MIP problems are NP-hard, exact methods (e.g., Branch-and-Bound, Branch-and-Cut, Branch-and-price and so on) are not suitable for solving large scale problems. Namely, obtaining exact solutions for majority of 0-1 MIP problems is not possible in a realistic or justifiable amount of time. More precisely, very often exact methods succeed in finding near-optimal solution quickly but need a lot of time to reach an optimal one. Additionally, even if they succeed in reaching an optimal solution quickly they sometimes consume a significant portion of the total solution time to prove its optimality. For these reasons, in many practical settings, exact methods are used as heuristics, stopping them before getting proof of optimality (e.g., imposing CPU time limit, maximum number of iterations to be performed, node limit etc.).

These drawbacks of exact methods have attracted researchers to develop many heuristic methods to tackle hard 0-1 MIP problems. The advantage of heuristics is not only their ability to produce high-quality solutions in short time, but also the fact that they can be easily combined with exact methods to speed them up (see Chapter 3). The early incumbent solutions produced by a heuristic can help a Branch-and-Bound algorithm to reduce the amount of memory needed to store the branch-and-bound tree as well as to accelerate the exploration of the branch-and-bound tree. So, this survey focuses on general heuristics for 0-1 MIP problems, not those that exploit the problem structure (i.e., lagrangian based heuristics (see e.g., [16, 130, 134, 136, 214]), or heuristics proposed for various special combinatorial structures, like scheduling and location problems, the traveling salesman problem etc. (see e.g., [71, 172, 188, 220])). Additionally, we do not review general heuristics for solving classes of problems that include 0-1 MIP problems as a special case such as heuristics for general MIP, convex integer programming, mixed integer non-linear programming and so on (for surveys of such heuristics we refer reader to [21, 26, 153] and references therein). More precisely, we strictly focus on the stand-alone heuristics for 0-1 MIP as well as those heuristics that use linear programming techniques or solve series of linear programming

models or reduced problems, deduced from the initial one, in order to produce a high quality solution. Our emphasis is on how mathematical programming techniques can be used for approximate problem solving, rather than on comparing performances of heuristics.

The rest of the chapter is organized as follows. In the next section we specify notation that will be used throughout the chapter and present standard LP basic solution representation. In section 1.3, we review heuristics that use pivot moves within the search for an optimal solution of the MIP in order to move from one extreme point to another. Section 1.4 contains the description of heuristics that use pseudo-cuts in order to cut-off portions of a solution space already examined in the previous solution process. Section 1.5 is devoted to so-called pump heuristics which purpose is to create a first feasible solution of the considered MIP. Section 1.6 provides overview of so-called proximity heuristics that seek a MIP feasible solution of a better quality in the proximity of the current incumbent solution. The next section entitled Advanced heuristics" is devoted to heuristics that may be considered as frameworks for building new heuristics for 0-1 MIP. Finally, the section 1.8 concludes the chapter and indicates possible directions for future work.

1.2 Notation and standard LP basic solution representation

Let α be a real number from the interval $[0, 1]$ and let $near(\alpha)$ refer to the nearest integer value of a real value $\alpha \in [0, 1]$ i.e., $near(\alpha) = \lfloor \alpha + 0.5 \rfloor$, where $\lfloor \alpha + 0.5 \rfloor$ represents the integer part of the number $\alpha + 0.5$. Further, let x be a vector such that $x_j \in [0, 1]$, $j \in I$. Then, $near(x)$ will represent the nearest integer (binary) vector relative to the vector x, whose each component is defined as $near(x)_j = near(x_j) = \lfloor x_j + 0.5 \rfloor$.

We first define a measure $u_r(\alpha)$ of integer infeasibility for assigning a value *alpha* to a variable by the following rule:

$$u_r(\alpha) = |\alpha - near(\alpha)|^r,$$

where exponent r is non negative number (e.g., between 0.5 to 2). Note that $|\alpha - near(\alpha)| = min\{\alpha - \lfloor\alpha\rfloor, \lceil\alpha\rceil - \alpha\}$. Obviously, such defined function takes value 0 if α is integer feasible, while otherwise it is strictly positive.

Starting from the previous definition, we may define a partial integer feasibility of a vector x relative to the subset $J \subset \mathcal{I}$ as:

$$u_r(x, J) = \sum_{j \in J} u_r(x_j).$$

The complement of the vector x, such that $x_j \in \{0, 1\},\ j \in \mathcal{I}$, relative to the subset $J \subset \mathcal{I}$ is the vector

$$x' = \overline{x(J)}$$

whose components are given as $x'_j = 1 - x_j$ for $j \in J$ and $x'_j = x_j$ for $j \notin J$.

Hamming distance between two solutions x and x' such that $x_j, x'_j \in \{0, 1\},\ j \in \mathcal{I}$ is defined by

$$\delta(x, x') = \sum_{j \in \mathcal{I}} |x_j - x'_j| = \sum_{j \in \mathcal{I}} x_j(1 - x'_j) + x'_j(1 - x_j).$$

The partial Hamming distance between x' and x', relative to the subset $J \subset \mathcal{I}$, is defined as $\delta(J, x, y) = \sum_{j \in J} \mid x_j - x'_j \mid$ (obviously, $\delta(\mathcal{I}, x, x') = \delta(x, x')$).

We denote by e the vector of all ones with appropriate dimension and by e_j the binary vector whose component j equals to 1, while all the other components are set to 0.

The MIP relaxation of the 0-1 MIP problem relative to a subset $J \subset \mathcal{I}$ is expressed as:

$$(MIP(J)) \begin{cases} maximize\ v = cx \\ s.t.\ Ax \leq b \\ 0 \leq x_j \leq U_j,\ j \in N = \{1, \ldots, n\} \\ x_j \in \{0, 1\},\ j \in J \subset \mathcal{I} \subseteq N \end{cases} \tag{1.2}$$

Given a MIP problem, P $\max\{cx \mid x \in X\}$ and an arbitrary LP feasible solution x^0. The problem *reduced* from the original problem P and associated with x^0 and a subset $J \subseteq \mathcal{I}$ is defined as:

$$P(x^0, J) \ \max\{cx \mid x \in X, x_j = x_j^0 \text{ for } j \in J \text{ such that } x_j^0 \in \{0,1\}\} \quad (1.3)$$

Note that in the case that $J = \mathcal{I}$, the reduced problem will be denoted by $P(x^0)$.

Similarly, given a MIP problem P and a solution $\widetilde{x}$ such that $\widetilde{x}_j \in \{0,1\},\ j \in \mathcal{I}$. Then, $MIP(P, \widetilde{x})$ will denote a minimization problem, obtained from MIP problem P by replacing the original objective function with $\delta(x, \widetilde{x})$:

$$MIP(P, \tilde{x}) \qquad \min\{\delta(\tilde{x}, x) \mid x \in X\}. \quad (1.4)$$

The LP relaxation of such defined MIP problem $MIP(P, \tilde{x})$ will be denoted by $LP(P, \tilde{x})$.

If C is a set of constraints, we will denote with $(P \mid C)$ the problem obtained by adding all constraints in C to the problem P.

It is well known that an optimal solution for the 0-1 MIP problem may be found at an extreme point of the LP feasible set, and special approaches integrating both cutting plane and search processes have been proposed to exploit this fact (Cabot and Hurter, 1968 [28]; Glover, 1968 [88]).

Property 1.2.1 *An optimal solution for the 0-1 MIP problem may be found at an extreme point of the LP feasible set.*

Proof. Since all continuous variables are slack variables in the MIP problem considered here, the MIP feasible set can be viewed as $\overline{X} = \{x \in \mathbb{R}^{|I|} | Ax \leq b; 0 \leq x_j \leq 1\}$. In other words, set $\overline{X}$ is intersection of a convex set $\{x \in \mathbb{R}^{|I|} | Ax \leq b\}$ and a cube $\{x \in \mathbb{R}^{|I|} | 0 \leq x_j \leq 1\}$. Therefore, since all 0-1 solutions are located at the extreme points of the unit cube, the all feasible 0-1 solutions are also located at at the extreme points of $\overline{X}$.□

The bounded simplex method proposed by Dantzig [57, 58] is an efficient method to solve the LP- relaxation of the MIP problem by systematically exploring extreme points of solution space. The search for an optimal extreme

point is performed by pivot operations, each of which moves from one extreme point to an adjacent extreme point by removing one variable from the current basis and bringing another variable (which is not in the current basis) into the basis. For our purposes, the procedure can be depicted in the following way. Suppose that the method is currently at some extreme point x^0 with corresponding basis B. The set of indices of all other variables (nonbasic variables) will be designated with $\overline{B} = N - B$. The extreme points adjacent to x^0 have the form

$$x^j = x^0 - \theta_j D_j \text{ for } j \in \overline{B} \tag{1.5}$$

where D_j is a vector associated with the nonbasic variable x_j, and θ_j is the change in the value of x_j that moves the current solution from x^0 to x^j along their connecting edge. The LP basis representation identifies the components D_{kj} of D_j, as follows

$$D_{kj} = \begin{cases} ((A^B)^{-1}A)_{kj} & \text{if } k \in B \\ \xi & \text{if } k = j \\ 0 & \text{if } k \in \overline{B} - \{j\} \end{cases} \tag{1.6}$$

where A^B represents matrix obtained from matrix A by selecting columns that correspond to the basic variables and $\xi \in \{-1, 1\}$. We choose the sign convention for entries of D_j that yields a coefficient for x_j of $D_{jj} = 1$ if x_j is currently at its lower bound at the vertex x^0, and of $D_{jj} = -1$ if x_j is currently at its upper bound at x^0.

Note that if we consider an extreme point x^0 and its adjacent extreme points x^j for $j \in \overline{B}$, we can conclude that the points x^j for $j \in \overline{B}$ are linearly independent and that point x^0 does not belong to the plane spanned by these points. Furthermore, this observation holds even when these θ_j values are replaced by any positive value. In what follows, the set of extreme points adjacent to an extreme point x^0 will be denoted by

$$\mathcal{N}_0(x^0) = \{x' = Pivot(x^0, p, q) : p \in B,\, q \in \overline{B}\}.$$

1.3 Pivot moves based heuristics

1.3.1 Pivot and Complement Heuristic

In 1980, Balas and Martin [8] proposed a *Pivot and Complement* (P&C) heuristic for a pure 0-1 MIP (i.e., $\mathcal{I} = N$), which relies on the fact that any 0–1 program may be considered as a linear program with the additional stipulation that all slack variables other than those in the upper bounding constraints must be basic. The P&C heuristic starts by solving the LP relaxation of the initial problem and then performs a sequence of pivots trying to put all slack variables into the basis while minimizing the objective function. Once a feasible solution is found, a local search is launched to improve it by complementing certain sets of 0–1 variables.

If solving LP relaxation of the initial problem does not yield feasible 0-1 solution, in order to find first MIP feasible solution, P&C heuristic performs pivoting, complementing as well as rounding and truncating in the way presented in Algorithm 1. More precisely, the P&C heuristic uses four types of neighborhood structures. Three neighborhood structures are based on pivot moves and one is based on complement moves:

- Neighborhood $\mathcal{N}_1$ is based on pivot moves that maintain primal feasibility of the LP relaxation while exchanging a nonbasic slack for a basic binary variable. The pivot occurs on nonbasic slack column $q \in (\overline{B} \cap \mathcal{S}) \setminus \mathcal{I}$ and a row $p \in B \cap \mathcal{I}$ for a basic binary variable such that

$$\mathcal{N}_1(x) = \{x' = Pivot(x, p, q) : q \in (\overline{B} \cap \mathcal{S}) \setminus \mathcal{I},\ p \in B \cap \mathcal{I}\}.$$

- Neighborhood $\mathcal{N}_2$ is based on pivot moves that also maintain primal feasibility of the LP relaxation, but the number of basic binary variables remains unchanged:

$$\mathcal{N}_2(x) = \{x' = Pivot(x, p, q) : (p, q) \in B \times \overline{B}, p, q \in \mathcal{S} \text{ or } p, q \in \mathcal{I} \text{ such that } u_r(x', \mathcal{I}) < u_r(x, \mathcal{I})\}.$$

More precisely, the pivots of this type exchange slack for a slack or a binary variable for a binary variable while reducing the sum of the integer infeasibilities.

- Neighborhood $\mathcal{N}_3$ is based on pivot moves that exchange a nonbasic slack for a basic binary variable violating primal feasibility. It is required that the slack variable must enter the basis with a positive value:

$$\mathcal{N}_3(x) = \{x' = Pivot(x, p, q) : (p, q) \in B \times (\overline{B} \cap \mathcal{S})\}.$$

- Neighborhood $\mathcal{N}_4^k$ is based on complement moves. Complementing of binary variables used in the first phase consists of flipping one or two variables at once. Complementing is preformed in order to reduce the infeasibility measured for a solution x as

$$\mu(x) = \sum_{i \in B \cap \mathcal{I}} max\{0, -x_i\} + \sum_{i \in B \cap \mathcal{I}} max\{0, x_i - 1\}.$$

A set J of nonbasic variables of size k is candidate for complementing if it minimize the infeasibility

$$\mathcal{N}_4^k(x) = \{\overline{x(J)} : J \subset \overline{B} \cap \mathcal{I}, |J| = k, A\overline{x(J)} \leq b, \mu(x) - \mu(\overline{x(J)}) > 0\}.$$

The procedure for finding the first feasible solution is executed iteratively. In each iteration firstly the search for a feasible solution is performed in the union of neighborhood structures $\mathcal{N}_1$ and $\mathcal{N}_2$. If the union is nonempty, the current incumbent solution is replaced by one from the union and the search is resumed starting from the new incumbent solution. The solution from the union that replaces incumbent solution is chosen according to the following rule. If neighborhood $\mathcal{N}_1$ of the incumbent solution is non-empty the best one, regarding the objective function value, is chosen from it, otherwise if neighborhood $\mathcal{N}_2$ of the incumbent solution is non-empty any solution from it is chosen. The search in the union $\mathcal{N}_1 \cup \mathcal{N}_2$ of the current incumbent solution is terminated either if the union is empty or the current incumbent solution is MIP feasible. If the obtained solution is MIP feasible or its rounding or

truncating yields the feasible MIP solution the overall procedure is stopped. Otherwise, the procedure selects a solution from the neighborhood $\mathcal{N}_3$ of the current incumbent solution that deteriorates primal feasibility as little as possible. After that the search is continued in the union of neighborhoods $\mathcal{N}_4^1$ and $\mathcal{N}_4^2$ in order to possible restore primal feasibility. If procedure succeeds to repair primal feasibility, the obtained solution is used as the input for the next iteration or the overall procedure is stopped. Stopping occurs if the obtained solution or any of solutions obtained by its rounding and truncating is MIP feasible. On the other hand, if procedure does not succeed to repair

primal feasibility, the failure of the procedure occurs.

Algorithm 1: Finding a feasible solution $FS()$

Function `FS()`;
Solve LP relaxation to obtain an optimal LP basic solution $\overline{x}$;
if $\overline{x} \in \{0,1\}^n$ **then**
  $Stop = True$; // it is optimal;
else
  Set $x = \overline{x}$;$Stop = False$;
end
while $Stop = False$ **do**
  while $\mathcal{N}_1(x) \cup \mathcal{N}_2(x) \neq \emptyset$ *and* $x \notin \{0,1\}^n$ **do**
    if $\mathcal{N}_1(x) \neq \emptyset$ **then**
      select $x' = argmax\{cx : x \in \mathcal{N}_1(x)\}$;
    else
      if $\mathcal{N}_2(x) \neq \emptyset$ **then**
        select $x' \in \mathcal{N}_2(x)$;
      end
    end
    set $x = x'$;
  end
  if $\{x, near(x), \lfloor x \rfloor\} \cap X \neq \emptyset$ **then**
    $Stop = True$; **Break**;
  end
  select $x' = argmin\{\mu(y) : y \in \mathcal{N}_3(x)\}$; set $x = x'$;
  while $\mathcal{N}_4^1(x) \cup \mathcal{N}_4^2(x) \neq \emptyset$ *and* x *infeasible* **do**
    if $\mathcal{N}_4^1(x) \neq \emptyset$ **then**
      select $x' = argmin\{\mu(y) : y \in \mathcal{N}_4^1(x)\}$
    else
      if $\mathcal{N}_4^2(x) \neq \emptyset$ **then**
        select $x' \in \mathcal{N}_4^2(x)$
      end
    end
    set $x = x'$;
  end
  if x *infeasible* **then**
    $Stop = Fail$;
  else
    if $\{x, near(x), \lfloor x \rfloor\} \cap X \neq \emptyset$ **then**
      $Stop = True$;
    end
  end
end

If Pivot and Complement heuristic succeeds to find a MIP feasible solution, an improvement phase is launched to possible improve the obtained solution. The improvement phase Algorithm 2 used inside Pivot and Complement heuristic is based on the variable fixing and complementing. Note that the improvement phase may be seen as a variable neighborhood descent approach which will be described in Chapter 2. The Improvement Phase firstly attempts to fix as many binary variables at their optimal values as possible. The choice of variables to be fixed is made according to the following rule. If the reduced cost of a nonbasic binary variable equals or exceeds the gap between the current lower and upper bounds on the objective function value, then its current value is the optimal and thus the variable is fixed. Complementing used in the improvement phase consists of complementing one, two or three variables at once. The set of variables to be flipped is determined in order to improve the current objective function value as follows. Let x be a current solution, then a set $J \subset \mathcal{I}$ of variables, is a candidate for complementing if

$$\sum_{j \in J}(1 - 2x_j)c_j \geq 1 \tag{1.7}$$

and

$$\sum_{j \in J}(1 - 2x_j)a_{ij} \leq b_i - \sum_{j \notin J} x_j a_{ij}, \forall i \in \{1, \ldots m\}. \tag{1.8}$$

In other words, in the improvement phase, the neighborhood structures

$$\mathcal{N}'^{k}_{4}(x) = \{\overline{x(J)} : J \subset \mathcal{I}, |J| = k, A\overline{x(J)} \leq b \text{ and } J \text{ satisfy (1.7) and (1.8)}\}.$$

are explored.

Algorithm 2: Improving a feasible solution x

Function LS(x);

1 $Stop = False$;
2 **while** $Stop = False$ **do**
3 Fix all 0-1 values in x that can be fixed;
4 **if** $\mathcal{N'}_4^1(x) \neq \emptyset$ **then**
select $x' = argmax\{cx : x \in \mathcal{N'}_4^1(x)\}$;
set $x = x'$;
continue;
end
5 **if** $\mathcal{N'}_4^2(x) \neq \emptyset$ **then**
select $x' \in \mathcal{N'}_4^2(x)$;
set $x = x'$;
continue;
end
6 **if** $\mathcal{N'}_4^3(x) \neq \emptyset$ **then**
select $x' \in \mathcal{N'}_4^3(x)$;
set $x = x'$;
continue;
end
$Stop = True$;
end

In 1989, Aboudi et al. [1] proposed an *enhancement of Pivot and Complement* heuristic by adding a objective function value constraint. In that way they obtained a new heuristic able to provide better solutions in shorter time than a basic Pivot and Complement heuristic.

In 2004, Balas et al. [12] proposed an extension of the Pivot and Complement heuristic called *Pivot and Shift* for mixed integer programs. The Pivot and Shift heuristic consists of two phases: a search phase that aims to find an integer feasible solution and an improvement phase that attempts to improve the solution returned by the search phase. In the search phase

a feasible solution is possible constructed by examining three neighborhood structures based on the pivot moves and one inspired by the local branching. If no replacement occurs searching one of the neighborhoods the search for a feasible solution is resumed exploring the small neighborhood of the current solution. The small neighborhood is created around the partial solution defined by those variables whose values are close to an integer. Specifically, let x be the current LP feasible solution, the search is done using MIP solver in the neighborhood defined as follows:

$$\mathcal{N}_5(x) = \{x' \in X : |\sum_{j \in J}(x'_j - near(x_j))| \leq 1\}$$

where $J = \{j \in B \cap \mathcal{I} : u_1(x_j) \leq \beta\}$ with β chosen to be a small positive value, e.g., 0.1.

If the search phase returns a MIP feasible solution, this solution is improved further in the improvement phase. The procedure tries to improve the current solution value by shifting some of the nonbasic integer-constrained variables up or down. Note that shifting the binary nonbasic variables, actually represents their complementing. Besides shifting one nonbasic variable, the procedure examines simultaneous shifting of two or three nonbasic variables. The variable or the set of variables to be shifted is determined as one that improve the objective function value while keeping the solution feasible. As soon as an improving shift is detected it is executed and the search is continued. The whole process is repeated as long as there is an improving shift. In other words, at each stage of shifting phase a neighborhood structure defined as:

$$\mathcal{N}_6^k(x) = \{x' \in X : J \subseteq \overline{B} \cap \mathcal{I},\ |J| = k; \delta(x, x') = \delta(J, x, x') = k;\ cx' > cx\}$$

is explored. As soon as shifting of variables is finished obtaining some solution x, that solution is further improved executing a large neighborhood search. A large neighborhood search consists of the exploration of neighborhood structure defined as

$$\mathcal{N}_7(x) = \{x' \in X : |\sum_{j \in \mathcal{I}}(x'_j - x_j)| \leq k\},$$

where k is a parameter, using the MIP solver. After, exhaustive testing the authors detected that the most suitable value for the parameter k is five.

1.3.2 Pivot and Tabu Heuristic

Tabu search (TS) [92] is a metaheuristic for solving optimization problems. It has its origins in heuristics based on surrogate constraint methods and cutting plane approaches that systematically violate feasibility conditions (Glover, 1977) [90] and a steepest ascent / mildest descent formulation developed by Pierre Hansen (1986) [115].

Since an optimal solution for the 0-1 MIP problem may be found at an extreme point of the LP feasible set (see Proposition 1.2.1), Glover and Lokketangen [154] proposed a Tabu Search based heuristic (Algorithm 3) for solving 0–1 MIP that exploits this fact. The procedure iteratively moves from one extreme point to an adjacent extreme point by executing a pivot move. Each pivot move is assigned a tabu status and a merit figure expressed as a function of the integer infeasibility and the objective function value. The move to be executed is then chosen as one that has the highest evaluation from those in the candidate set. During the search for a pivot move to be preformed the current best integer solution x^* is updated as soon as some

better solution is encountered.

Algorithm 3: Tabu search for 0-1 MIP

Function `TS`();

1 Solve the LP relaxation to obtain an optimal LP basic solution $\overline{x}$;

2 **if** $\overline{x}$ *MIP feasible* **then**

  return $\overline{x}$;

end

3 Set $x = \overline{x}$;

4 $v^* = -\infty$;

while *Stopping criterion is not satisfied* **do**

5 Consider the neighborhood of x that contains feasible pivot moves that lead to adjacent basic LP feasible solutions;

6 If a candidate move would lead to an 0-1 MIP feasible solution x' such that $cx' > v^*$, record x as the new x^* and set $v^* = cx'$;

7 Select the pivot move with the highest move evaluation, applying tabu restrictions and aspiration criteria;

8 Execute the selected pivot, updating the associated tabu search memory and guidance structures;

end

Note that the method may not necessarily visit the best MIP feasible neighbor of the current solution, since the move evaluation of Step 2 depends on other factors in addition to the objective function value (see below).

Obviously, three elements are required to implement Tabu Search procedure for solving 0–1 MIP:

1. neighborhood structure (the candidate list of moves) to examine;
2. the function for evaluating the moves;
3. the determination of rules (and associated memory structures) that define tabu status.

- **Neighborhood structure:** Let x^0 denote the current extreme point with the set of basic variables B. The neighborhood structure explored

by Tabu search which contains extreme points adjacent to the given extreme point x^0, is defined as:

$$\begin{aligned}\mathcal{N}_0(x^0) = \quad & \{x' = Pivot(x^0, p, q) : q \in \overline{B},\, p \in B\} = \\ & \{x^h = x^0 - D_h\theta_h \,\text{for}\, h \in \overline{B}\}\end{aligned}$$

where D_h is a vector associated with the nonbasic variable x_h, and θ_h is the change in the value of x_h that moves the current solution from x^0 to x^h along their connecting edge (see Section 1.2). We thus start the search from an integer infeasible point, and may also spend large parts of the search visiting integer infeasible solution states. However, for large problem examination of entire neighborhood is not possible in a reasonable amount of time. Therefore, the strategies described in Glover et al. (1993) [104] and in Glover (1995) [94] are used in order to reduce neighborhood size.

- **Move evaluation:** The move evaluation function is composite, based on two independent measures. The first measure is the change in objective function value when going from x^0 to $x^h \in \mathcal{N}_0(x^0)$, and the second measure is the change in integer infeasibility. Restricting consideration to $h \in \bar{B}$, we define

$$\Delta v(h) = cx^h - cx^0$$

$$\Delta u(h) = u_r(x^0, \mathcal{I}) - u_r(x^h, \mathcal{I}).$$

Note that it is not necessary to execute a pivot to identify x^h or the values of $u_r(x^h, \mathcal{I})$ and cx^h, since only the vector D_h, the scalar θ_h, and the current solution x^0 are required to make this determination.

Overall procedure for determining the best solution works in the following way. Firstly, all solutions are classified in four groups according to the sign of the change in the objective value $\Delta v(h)$ and the change in the integer infeasibility $\Delta u(h)$ that would occur replacing current solution by one neighboring. After that the best solution is determined

using some of the following tests:

- **Weighted sum:** To each solution x^h, the value $\Delta(h) = \Delta v(h) + \Delta u(h)$ is assigned. After that the first non tabu solution with the highest value of $\Delta(h)$ is accepted.
- **Ratio test:** For each solution type, of 4 aforementioned, ratio of $\Delta v(h)$ and $\Delta u(h)$ is calculated, firstly. After that the best solution of each type is determined, and finally the best one among them, using the specific rules (see Glover et al.[154]), is accepted.
- **Weighted sum solution evaluation, sorted by solution type:** The solutions are evaluated as a weighted sum (i.e., $\Delta(h) = \alpha\Delta v(h) + \beta\Delta u(h)$), but first sorted according to solution type, and then according to the solution evaluation within each solution type group. To determine the best solution to accept the same rule as for the ratio test is used.
- **Ratio test favoring integer feasibility:** This test is intended to drive the search more strongly to achieve integer feasibility than the basic ratio test. Therefore it gives priority to solutions that reduce the integer infeasibility.

- **Tabu status:** The tabu status is created according to two tabu records, $Recency(j)$ and $Frequency(j)$ for each variable x_j, $j \in N$. $Recency(j)$ is used to record recency information, while $Frequency(j)$ to measure the number of iterations that x_j has been basic. At the beginning $Recency(j)$ is set to a large negative number and then, whenever x_j becomes nonbasic, the value of $Recency(j)$ is set to the number of iteration at which that change occurs. If at some iteration $Tabu(j)$ value is changed the status of variable x_j is set to Tabu for a predefined number of iterations. Similarly, a nonbasic variable x_j is penalized in order not to be chosen to become basic according to the value either of $Frequency(j)$ or $Frequency(j)/Current\_Iteration$.

1.3.3 Pivot Cut and Dive Heuristic

In 2007, Eckstein and Nediak [65] presented a four layered heuristic called *Pivot Cut and Dive* (Algorithm 4) for pure 0–1 MIP programming problems. In the first layer gradient-based pivoting is applied, built around a concave merit function that is zero at integer-feasible points and positive elsewhere in the unit cube (noting $x_j \in \{0,1\}$ is equivalent to $x_j(1-x_j)=0$). The general form of such merit function is given in the following way. Consider a collection of continuously differentiable concave functions $\phi_i : \mathbb{R} \to \mathbb{R}$, $i \in \mathcal{I}$, such that $\phi_i(0) = \phi_i(1) = 0$ and $\phi_i(x) > 0$ for all $x \in]0,1[$. Then a concave merit function has a form $\psi(x) = \sum_{i\in\mathcal{I}} \phi_i(x_i)$.

Let the reduced costs for any cost vector t be denoted by $z(t)$. Then for previously defined merit function the following statements holds:

Property 1.3.1 *For a given LP basic solution x^0 and feasible direction D_j, $j \in \bar{B}$ from x^0 holds $z(\nabla\psi(x^0)) = \nabla\psi(x^0)D_j$. Additionally, from the concavity of $\psi(\cdot)$ and its differentiability at x^0, we have $\psi(x^h) \leq \psi(x^0) + \theta_h z_h(\nabla\psi(x^0))$, $h \in \overline{B}$. Moreover, for linear ψ, the last relation holds with equality and therefore for a fixed vector $f \in \mathbb{R}^n$ with $fD_j \neq 0$,we have*

$$\frac{\psi(x^j) - \psi(x^0)}{fx^j - fx^0} \leq \frac{z_j(\nabla\psi(x^0))}{z_j(f)}.$$

Based on this proposition, the procedure attempts to round a fractional solution (LP solution) via primal simplex pivots deteriorating its objective function value as less as possible. In order to achieve that three types of pivots are applied in a sequence after exhausting pivots of the previous type:

- **Pivot 1.** Pivots that decrease the merit function but do not decrease the objective function value. These pivots define the following neighborhood structure:

  $$\mathcal{N}''_1(x) = \{x' = Pivot(x,p,q) : q \in \overline{B},\ p \in B,\ \psi(x') < \psi(x),\ cx' \geq cx\}$$

- **Pivot 2.** Pivots that locally improves the merit function, at the least possible cost in terms of the objective function. Using these pivots the

following neighborhood structure is explored:

$$\mathcal{N}''_2(x) = \{x' = Pivot(x, p, q) : q \in \overline{B},\ p \in B,\ z_q(\nabla\psi(x^0)) < 0\}$$

- **Pivot 3. (probing layer)** In probing phase possible pivots are explicitly tested until a satisfactory improving pivot is found or the list of possible entering variables is exhausted. In order not to spend a lot of time, inspecting is performed according to the list that gives priority to the candidate entering variables x^j that has low values of $z_j(\nabla\psi(x^0))$ and $z_j(c)$. Before accepting some pivot move, the new iterate x that would result is computed and the checking whether is $\psi(x) < \psi(x^0)$ or not is performed. If $\psi(x) \geq \psi(x^0)$ we abandon the pivot and proceed to the next in the list. On the other hand, if the pivot passes this test, the objective sacrifice rate is calculated as :

$$\frac{cx - cx^0}{\psi(x^0) - \psi(x)}.$$

  If this sacrifice rate is acceptable in relation to the prior history then x is accepted as the next iterate and no further pivots are probed. Otherwise, probing continues. If none of pivots was accepted, the rounding process fails. Note that if pivot 3 is executed the resulting solution will belong to the neighborhood

$$\mathcal{N}''_3(x) = \{x' = Pivot(x, p, q) : q \in \overline{B},\ p \in B,\ \psi(x') < \psi(x)\}.$$

In the case of the failure of the probing layer a convexity cut violated by the current vertex and all adjacent vertices is generated (third layer). If the problem obtained adding previous cut is feasible, the rounding procedure is repeated. However, if the probing fails, and the resulting convexity cut appears excessively shallow the final layer of the heuristic: a recursive, depth-first diving is applied in order to possible recover feasibility. If the feasibility is successfully repaired the rounding procedure is repeated, while otherwise,

the procedure fails (no feasible solution of (sub)problem is found).

Algorithm 4: Pivot, cut and dive heuristic for 0-1 MIP

Function `Pivot&cut&dive`(P);
Solve the LP relaxation of P to obtain an optimal LP basic solution $\overline{x}$;
$Stop = False$;
$Integer = False$;
Set $x = \overline{x}$;
while *Stop=False and Integer = False* **do**
  if *x MIP feasible* **then**
    $Integer = True$;
  end
  if $\mathcal{N}''_1(x) \neq \emptyset$ **then**
    select best $x' \in \mathcal{N}''_1(x)$ according to imposed rule;
    set $x = x'$;
    continue;
  end
  if $\mathcal{N}''_2(x) \neq \emptyset$ **then**
    select best $x' \in \mathcal{N}''_2(x)$ according to imposed rule;
    set $x = x'$;
    continue;
  end
  if $\mathcal{N}''_3(x) \neq \emptyset$ **then**
    select first $x' \in \mathcal{N}''_3(x)$ according to imposed rule;
    set $x = x'$;
    continue;
  end
  $Stop = True$;
end
if *Integer=False* **then**
  $Q = (P|$ convexity cut$)$;
  if *Q is feasible* **then**
    `Pivot&cut&dive`(Q);
  else
    Try to repair feasibility of Q ;
    if *Feasibility repaired* **then**
      `Pivot&cut&dive`(Q);
    else
      Report failure;
    end
  end
end

1.4 Pseudo-cut based heuristics

1.4.1 Local branching heuristics

In 2003, Fischetti and Lodi [74] proposed a *Local Branching* (LB) heuristic for 0–1 MIP based on the soft variable fixing and the observation that the neighborhood of a feasible MIP–solution often contains solutions of possibly better qualities. They introduced a so-called branching constraint in order to define a neighborhood of a given feasible solution and a branching criterion within an enumerative scheme. Although the Local Branching heuristic had been conceived as an improvement heuristic for 0–1 MIP, the same authors showed in [75] that it could be used as a heuristic for building a first feasible solution.

So-called variable fixing or diving consists of eliminating integer variables by fixing them to some promising values and resolving the new LP problem, in order to determine the next fixing candidates. This procedure is iterated, and thereby the subproblems get smaller and smaller. Unfortunately, diving heuristics often end in an infeasible subproblem. This is because of the fact that it is difficult in advance to estimate which value to assign to some variable. So, a better option is to perform so-called soft fixing in which it is required that a certain number of variables, take the same values as in the incumbent solution, without fixing any of those variables. The Local Branching heuristic performs soft variable fixing adding a single linear constraint to the original problem. The added constraint $\delta(x, \tilde{x}) \leq k$ defines so-called $k - opt$ neighborhood of the incumbent solution $\tilde{x}$ that is stated as:

$$\mathcal{N}^k(\tilde{x}) = \{x \in X : \delta(x, \tilde{x}) \leq k\}.$$

Additionally, this constraint may be used as a branching strategy. In that case the branching rule would be $\delta(x, \tilde{x}) \leq k$ or $\delta(x, \tilde{x}) > k$. With this branching rule solution space will be divided into two parts. The part defined $\delta(x, \tilde{x}) \leq k$ may be relatively small with appropriately chosen k.

The LB algorithm (Algorithm 5) starts with the original formulation, and CPLEX is used to get a feasible solution, i.e., an initial solution $\tilde{x}$. Then the k-opt neighborhood of that solution is explored using CPLEX respecting the

predefined time limit. If a better solution $\tilde{x}'$ is found, branching is performed. New problem is generated reversing the constraint that defines the k-opt neighborhood of $\tilde{x}$ (i.e., explored part of solution space is excluded) and adding new branching constraint centered around the new incumbent $\tilde{x}'$ that defines its k-opt neighborhood. This branching procedure is iterated until there is no improvement in the objective function value. Additionally, the LB heuristic includes an intensification and a diversification phase. In the intensification phase if the solution is not improved by CPLEX within the imposed time limit, the size of neighborhood is reduced (for example, halved) and CPLEX is called again. On the other hand, in the diversification phase a new solution is generated for the next branching step. This solution is obtained as a feasible solution of the program obtained increasing the right-hand side value of the last added branching constraint (for example for $k/2$), adding new constraint $\delta(x, \tilde{x}) \geq 1$ and deleting all other branching constraints. This step is invoked either if CPLEX proves infeasibility or it does not find any feasible solution.

Algorithm 5: Local Branching for 0-1 MIP

Function LB(k_0);
Find an initial solution $\tilde{x}$ by CPLEX;
Set $k = k_0$;
Set $Y = X$;
repeat
  if *CPLEX finds* $\tilde{x}' \in Y \cap \mathcal{N}^k(\tilde{x})$ *better than* $\tilde{x}$ **then**
    $Y = Y - \mathcal{N}^k(\tilde{x})$;
    set $\tilde{x} = \tilde{x}'$;
  else
    if $Y \cap \mathcal{N}^k(\tilde{x}) \neq \emptyset$ **then**
      $k = \lfloor k/2 \rfloor$;
    else
      Set $k = k + \lfloor k/2 \rfloor$;
      $Y = \{x \in X : \delta(x, \tilde{x}) \geq 1\}$;
    end
  end
until *Stopping criterion is satisfied*;
return $\tilde{x}$;

In 2006, Hansen et al. [124] proposed variable neighborhood search (VNS)

heuristic combined with LB, called *Variable Neighborhood Branching*, for solving mixed-integer programs which may be seen as a generalization of Local Branching. The main advantage of the proposed VNS comparing to the LB heuristic is the fact that it performs more systematic neighborhood exploration than Local Branching.

1.4.2 Iterative heuristics based on relaxations and pseudo-cuts

In 2011, Hanafi and Wilbaut [112, 221], proposed several convergent heuristics for 0–1 MIP problems, consisting of generating two sequences of upper and lower bounds by solving LP or MIP relaxations and sub-problems (see Algorithm 6). The process continues until the established bounds guarantee that no better solution solution could be found. Unfortunately, in practice all of these heuristics turned out to be very slow and therefore the authors used them as heuristics with a maximum number of iterations.

Algorithm 6: Framework for convergent heuristics for 0–1 MIP problems

Function `Convergent_heuristic`(P);
1 Set $Q = P$;
2 Choose a relaxation R of Q;
3 **repeat**
4 Solve the relaxation R of Q to obtain an optimal solution $\bar{x}$; //Lower bound
5 Generate a solution x^0, solving the reduced problem of Q associated with the solution $\bar{x}$; //Upper bound
6 **if** x^0 *better than* x^* **then**
7 Set $x^* = x^0$;
end
8 Add pseudo cut(s) to Q in order to exclude already generated solution x^0;
until *optimality is proven or Stopping criterion is satisfied*;
return x^*;

Hanafi and Wilbaut proposed several variants of a convergent heuristic [112, 221]:

- **Linear programming-based algorithm (LPA).** At each iteration, the LPA algorithm solves the LP-relaxation of the current problem Q to generate an optimal solution $\bar{x}$. After that the reduced problem $P(\bar{x})$

is generated from the initial problem P by setting the 0-1 variables to their value in the solution $\bar{x}$ if these variables are integers. Then the associated reduced problem $P(\bar{x})$ is solved exactly to generate a feasible solution x^0 for the original problem P. If the current best feasible solution x^* is not optimal, the current problem Q is enriched by a pseudo-cut to avoid generating the optimal basis of the LP-relaxation more than once. The process stops if the difference between the upper and the lower bounds is less than 1, i.e., if the condition $c\bar{x} - cx^* < 1$ is satisfied.

- **Mixed integer programming-based algorithm (MIPA).** This algorithm is derived from LPA algorithm solving MIP relaxation of the current problem Q, instead of solving its LP relaxation. In the first iteration of the algorithm, the mixed integer programming relaxation is defined from an optimal solution of the LP-relaxation forcing the fractional variables of the solution of the LP relaxation to be integers in the next iteration. Then the fractional variables in an optimal solution of the current MIP-relaxation are constrained to be integers in the next iteration.

- **Iterative relaxation-based heuristic (IRH).** At each iteration IRH heuristic solves LP relaxations of the current problem Q and obtains an optimal solution $\bar{x}$. After that it finds an optimal solution, $\tilde{x}$ of MIP relaxation based on the solution $\bar{x}$. In the next step, two reduced problem $P(\bar{x})$ and $P(\tilde{x})$ are solved and therefore two pseudo-cuts are added to the problem Q. Whole process is repeated predefined number of iterations or until proving the optimality of the current best feasible solution.

- **Iterative Independent relaxation-based heuristic (IIRH).** IIRH requires an initial phase in order to define the first MIP-relaxation as in the MIPA. After this initial phase, the LPA and the MIPA are applied simultaneously. The best lower and upper bounds generated during the process are then memorized.

1.4.3 Hybrid Variable Neighborhood decomposition search heuristics

In 2010, Lazic et al. [151] proposed a hybrid heuristic for solving 0-1 mixed integer programs which combines variable neighborhood decomposition search (VNDS) with the CPLEX MIP solver (see Algorithm 8). The algorithm starts solving the LP-relaxation of the original problem obtaining an optimal solution $\bar{x}$. If the optimal solution $\bar{x}$ is integer feasible the procedure returns $\bar{x}$ as an optimal solution of the initial problem. Otherwise, an initial feasible solution x is generated. At each iteration of the VNDS procedure, the distances $\delta_j = |x_j - \bar{x}_j|$ between the current incumbent solution values and corresponding LP-relaxation solution values are computed. Those distance values serve as criteria of choosing variables that will be fixed. Namely, at each iteration k variables which indices correspond to the indices of k smallest δ_j values, are fixed to values at their values in the current incumbent solution x. After that the resulting problem is solved using the CPLEX MIP solver. If an improvement of the current solution is achieved, a Variable Neighborhood Descent branching is launched as the local search in the whole solution space and the process is repeated. If not, the number of fixed variables in the current subproblem is decreased. The pseudo-code is given in Algorithm 8. The input parameters for the VNDS algorithm are: the MIP problem P; the parameter d, which controls the change of neighbourhood size during the search process; parameters $t_{max}, t_{sub}, t_{vnd}, t_{mip}, r_{max}$ which represent the maximum running time allowed for VNDS, time allowed for solving subproblems, time allowed for call to the `VND-MIP` procedure, time allowed for call to the MIP solver within the `VND-MIP` procedure, respectively. Finally, the parameter r_{max} represents maximum size of neighbourhood to be explored within the `VND-MIP` procedure. In the pseudo-code the statement of the form $y = \texttt{FindFirstFeasible}(P)$ denotes a call to a generic MIP solver, an attempt to find a first feasible solution of an input problem P. Further, the statement of the form $y = \texttt{MIPsolve}(P, t, x^*)$ denotes a call to a generic MIP solver to solve input problem P within a given time limit t starting from

the the best solution found x^*.

Algorithm 7: Variable Neighborhood Descent branching

Function VNDS($P, t_{vnd}, t_{mip}, r_{max}, x'$);
Set $r = 1$, $t_{start} = CpuTime()$, $t = 0$;
Set $Q = P$;
while $t < t_{vnd}$ *and* $r \leq r_{max}$ **do**
    set $time\_limit = min\{t_{mip}, t_{vnd} - t\}$;
    $Q = (Q|\{\delta(x', x) \leq r\})$;
    $x'' =$ MIPsolve($Q, time\_limit, x'$);
    switch *solution status* **do**
        case OptSolFound:
            Reverse last pseudo-cut into $\delta(x', x) > r + 1$;
            $x' = x''$, $r = 1$;
        case feasibleSolFound:
            Replace last pseudo-cut with $\delta(x', x) \geq 1$;
            $x' = x''$, $r = 1$;
        case ProvenInfeasible:
            Reverse last pseudo-cut into $\delta(x', x) > r + 1$;
            $r = r + 1$;
        case nofeasiblesolfound:
            return x'';
    endsw
    set $t_{end} = CpuTime()$, $t = t_{end} - t_{begin}$;
end
return x'';

Algorithm 8: Variable neighborhood decomposition search heuristic for 0-1 MIP

Function VNDS($P, d, t_{max}, t_{sub}, t_{vnd}, t_{mip}, r_{max}$);
Solve the LP relaxation of P to obtain an optimal LP basic solution $\overline{x}$;
if $\overline{x}$ MIP feasible **then return** $\overline{x}$;
$x^* =$ FindFirstFeasible(P);
set $t_{start} = CpuTime(); t = 0$;
while $t < t_{max}$ **do**
  $\delta_j =\mid x_j - \overline{x}_j \mid$; index x_j so that $\delta_j \leq \delta_{j+1}$, $j = 1, \ldots, |\mathcal{I}| - 1$;
  set $q =\mid \{j \in \mathcal{I} \mid \delta_j \neq 0\} \mid$;
  set $k_{step} = near(q/d)$, $k = p - k_{step}$;
  while $t < t_{max}$ *and* $k > 0$ **do**
    $x' =$ MIPsolve($P(\tilde{x}, \{1, \ldots, k\}), t_{sub}, x^*$) ;
    if $cx' > cx^*$ **then**
      $x =$ VND-MIP($P, t_{vnd}, t_{mip}, r_{max}, x'$);
      break;
    else
      if $k - k_{step} > p - q$ **then** $k_{step} = \max\{near(k/2), 1\}$;
      set $k = k - k_{step}$;
      set $t_{end} = CpuTime()$, $t = t_{end} - t_{begin}$;
    end
  end
end
return x^*;

In 2010, Hanafi et al. [114] proposed a hybrid Variable Neighborhood decomposition search heuristic that constitutes an improved version of Variable Neighborhood decomposition search heuristic proposed in [151]. The enhancement is achieved by restricting the search space by adding pseudo cuts, in order to avoid multiple explorations of the same areas. A sequence of lower and upper bounds on the problem objective is produced by adding pseudo-cuts, thereby reducing the integrality gap.

1.5 Pump heuristics

1.5.1 Feasibility Pump

The Feasibility Pump (FP) heuristic (Algorithm 9) was proposed by Fischetti et al. in [77]. The proposed heuristic turned out to be very efficient in finding a feasible solution to 0-1 MIP. The work of the FP heuristic may be outlined as follows. Starting from an optimal solution of the LP-relaxation, the FP heuristic generates two sequences of solutions $\overline{x}$ and $\tilde{x}$, which satisfy LP-feasibility and integrality feasibility, respectively. These sequences are built iteratively. At each iteration, a new binary solution $\tilde{x}$ is obtained from the fractional $\overline{x}$ by simply rounding its integer-constrained components to the nearest integer, i.e., $\tilde{x} = near(\overline{x})$, while a new fractional solution $\overline{x}$ is defined as an optimal solution of the $LP(MIP, \tilde{x})$ problem, i.e.,:

$$\min\{\delta(x, \tilde{x}) | Ax \leq b,\ 0 \leq x_j \leq U_j,\ j \in N\}. \tag{1.9}$$

Thus, a new fractional solution $\overline{x}$ is generated as the closest feasible LP solution with respect to the solution $\tilde{x}$. However, after a certain number of iterations the FP procedure may start to cycle, i.e., a same sequence of points $\overline{x}$ and $\tilde{x}$ is visited again and again. That issue is resolved applying a random perturbation move of the current solution $\tilde{x}$ as soon as cycle is detected. In the original implementation, that is performed flipping a random number $t \in [T/2, 3T/2]$ entries $\tilde{x}_j,\ j \in \mathcal{I}$ with the highest value $|\overline{x}_j - \tilde{x}_j|$, where T is predefined parameter. The procedure finishes its work as soon as a feasible solution is detected, or some of stopping criteria are fulfilled. The stopping criteria usually contain a running time limit and/or the total

number of iterations.

Algorithm 9: Feasibility Pump for 0-1 MIP

Function FP(P, T);
Solve the LP relaxation of P to obtain an optimal LP basic solution $\overline{x}$;
repeat
  $\tilde{x} = near(\overline{x})$;
  Solve the $LP(P, \tilde{x})$ problem to obtain an optimal solution $\overline{x}$;
  if *cycle detected* **then**
    choose a random number $t \in [T/2, 3T/2]$;
    flip values of t variables with the highest values $|\overline{x}_j - \tilde{x}_j|$;
  end
until $\overline{x}$ *is MIP feasible or stopping criterion is satisfied*;
return $\overline{x}$;

This basic Feasibility Pump was extended to the general feasibility pump, a heuristic for general mixed-integer problems [18]. The general feasibility pump employs the distance function in which the general integer variables also contribute to the distance. On the other hand, in order to enhance FP so that it returns a good-quality initial solution, so called *Objective feasibility pump* was proposed in [2]. The idea of the objective FP is to include the original objective function as a part of the objective function of the problem considered at a certain pumping cycle of FP. At each pumping cycle, the actual objective function is computed as a linear combination of the feasibility measure and the original objective function. Results reported in [2] indicate that this approach usually yields considerably higher-quality solutions than the basic FP. However, it generally requires much longer computational time.

1.5.2 Variable Neighborhood Pump

In 2010, Hanafi et al. [113] proposed a new method for finding an initial feasible solution for Mixed integer programs called *Variable Neighborhood Pump* (VNP) (Algorithm 10), that combines Variable neighborhood branching (VNB) [124] and Feasibility pump heuristics [77]. The VNP works in the following way. Firstly, an optimal solution $\overline{x}$ of the LP-relaxation of the initial 0-1 MIP problem is determined. After that, the obtained solution is rounded

and on it is applied one iteration of the FP pumping cycle in order to obtain a near-feasible vector $\tilde{x}$. Then on the solution $\tilde{x}$, variable neighbourhood branching, adapted for 0-1 MIP feasibility [113], is applied, in an attempt to locate a feasible solution of the original problem. If VNB does not return a feasible solution a pseudo-cut is added to the current subproblem in order to change the linear relaxation solution, and the process is iterated. VNB returns either a feasible solution or reports failure and returns the last integer (infeasible) solution.

Algorithm 10: Variable Neighborhood Pump for 0-1 MIP

Function `VNP`(P);
Set $proceed1 =$ `true`;
while $proceed1$ **do**
  Solve the LP relaxation of P to obtain an optimal LP basic solution $\overline{x}$;
  Set $\tilde{x} = near(\overline{x})$;
  Set $proceed2 =$ `true`;
  while $proceed2$ **do**
    if $\overline{x}$ *is integer* **then return** $\overline{x}$;
    Solve the $LP(P, \tilde{x})$ problem to obtain an optimal solution $\overline{x}$;
    if $\tilde{x} \neq near(\overline{x})$ **then**
      $\tilde{x} = near(\overline{x})$;
    else
      Set $proceed2 =$ `false`;
    end
  end
  $k_{min} = \lfloor \delta(\tilde{x}, \overline{x}) \rfloor$; $k_{max} = \lfloor (|\mathcal{I}| - k_{min})/2 \rfloor$; $k_{step} = (k_{max} - k_{min})/5$;
  $x' =$ `VNB`$(P, \tilde{x}, k_{min}, k_{step}, k_{max})$;
  if $x' = \tilde{x}$ **then**
    $P = (P \mid \delta(x, \overline{x}) \geq k_{min})$; Update $proceed1$;
  else
    return x';
  end
end
Output message: "No feasible solution found"; **return** $\tilde{x}$;

1.5.3 Diving heuristics

In 2014, Lazic et al. [152] proposed two diving heuristics for obtaining a first MIP feasible solution. Diving heuristics are based on the systematic hard variable fixing (diving) process, according to the information obtained from the linear relaxation solution of the problem. They rely on the observation that a general-purpose MIP solver can be used not only for finding (near) optimal solutions of a given input problem, but also for finding the initial feasible solution.

The variable neighbourhood (VN) diving algorithm begins by obtaining the LP-relaxation solution $\overline{x}$ of the original problem P and generating an initial integer (not necessarily feasible) solution $\tilde{x} = near(\overline{x})$ by rounding the LP-solution $\overline{x}$. If the optimal solution $\overline{x}$ is integer feasible for P, VN diving stops and returns $\overline{x}$. At each iteration of the VN diving procedure, the distances $\delta_j =\mid \tilde{x}_j - \overline{x}_j \mid$ from the current integer solution values $(\tilde{x}_j)_{j\in\mathcal{I}}$ to the corresponding LP-relaxation solution values $(\overline{x}_j)_{j\in\mathcal{I}}$ are computed and the variables $\tilde{x}_j, j \in \mathcal{I}$ are indexed so that $\delta_1 \leq \delta_2 \leq \ldots \leq \delta_{|\mathcal{I}|}$. Then, VN diving successively solve the subproblems $P(\tilde{x}, \{1, \ldots, k\})$ obtained from the original problem P, where the first k variables are fixed to their values in the current incumbent solution $\tilde{x}$. If a feasible solution is found by solving $P(\tilde{x}, \{1, \ldots, k\})$, it is returned as a feasible solution of the original problem P. Otherwise, a pseudo-cut $\delta(\{1, \ldots, k\}, \tilde{x}, x) \geq 1$ is added in order to avoid exploring the search space of $P(\tilde{x}, \{1, \ldots, k\})$ again, and the next subproblem is examined. If no feasible solution is detected after solving all subproblems $P(\tilde{x}, \{1, \ldots, k\})$, $k_{min} \leq k \leq k_{max}$, $k_{min} = k_{step}$, $k_{max} = |\mathcal{I}| - k_{step}$, the linear relaxation of the current problem P, which includes all the pseudo-cuts added during the search process, is solved and the process is iterated. If no feasible solution has been found due to the fulfillment of the stopping criteria, the algorithm reports failure and returns the last (infeasible) integer solution.

The pseudo-code of the VN diving heuristic is given in the Algorithm 11. The input parameters for the VN diving algorithm are the input MIP problem P and the parameter d, which controls the change of neighbourhood size during the search process. In the pseudo-code the statement of the form $y =$ `FindFirstFeasible`(P, t) denotes a call to a generic MIP solver, an

attempt to find a first feasible solution of an input problem P within a given time limit t. If a feasible solution is found, it is assigned to the variable y, otherwise y retains its previous value.

Algorithm 11: Variable neighbourhood diving for 0-1 MIP feasibility.

Function `VNdiving`(P, d);
1 Set *proceed*1 = `true`, *proceed*2 = `true`; Set *timeLimit* for subproblems;
2 **while** *proceed*1 **do**
3 Solve the LP relaxation of P to obtain an optimal LP basic solution $\overline{x}$;
4 $\tilde{x} = near(\overline{x})$;
5 **if** $\overline{x} = \tilde{x}$ **then return** $\tilde{x}$;
6 $\delta_j =\mid \tilde{x}_j - \overline{x}_j \mid$; index x_j so that $\delta_j \leq \delta_{j+1}$, $j = 1, \ldots, |\mathcal{I}| - 1$;
7 Set $n_d =\mid \{j \in \mathcal{I} \mid \delta_j \neq 0\} \mid$, $k_{step} = near(n_d/d)$, $k = |\mathcal{I}| - k_{step}$;
8 **while** *proceed*2 **and** $k \geq 0$ **do**
9 $J_k = \{1, \ldots, k\}$; $x' =$ `FindFirstFeasible`$(P(\tilde{x}, J_k), timeLimit)$;
10 **if** $P(\tilde{x}, J_k)$ *is proven infeasible* **then** $P = (P \mid \delta(J_k, \tilde{x}, x) \geq 1)$;
11 **if** x' is feasible **then return** x';
12 **if** $k - k_{step} > |\mathcal{I}| - n_d$ **then** $k_{step} = \max\{near(k/2), 1\}$;
13 Set $k = k - k_{step}$;
14 Update *proceed*2;
end
15 Update *proceed*1;
end
16 Output message: "No feasible solution found"; **return** $\tilde{x}$;

In the case of variable neighbourhood diving, a set of subproblems $P(\tilde{x}, J_k)$, for different values of k, is examined in each iteration until a feasible solution is found. In the single neighbourhood diving procedure, we only examine one subproblem $P(\tilde{x}, J_k)$ in each iteration (a single neighbourhood, see Algorithm 12). However, because only a single neighbourhood is examined, additional diversification mechanisms are required. This diversification is provided through keeping the list of constraints which ensures that the same reference integer solution $\widetilde{x}$ cannot occur more than once (i.e., in more than one iteration) in the solution process. An additional MIP problem Q is introduced to store these constraints. In the beginning of the algorithm, Q is initialized as an empty problem (see line 4 in Algorithm 12). Then, in each iteration, if the current reference solution $\widetilde{x}$ is not feasible (see line 8 in Algorithm 12), constraint $\delta(\widetilde{x}, x) \geq \lceil \delta(\widetilde{x}, \overline{x}) \rceil$ is added to Q (line 9). This guarantees that

future reference solutions can not be the same as the current one, since the next reference solution is obtained by solving the problem MIP$(Q, near(\overline{x}))$ (see line 17), which contains all constraints from Q, (see definition (3.9)). The variables to be fixed in the current subproblem are chosen among those which have the same value as in the linear relaxation solution of the modified problem LP$(\widetilde{x})$, where $\widetilde{x}$ is the current reference integer solution (see lines 7 and 11). The number of variables to be fixed is controlled by the parameter α (line 11). After initialization (line 5), the value of α is updated in each iteration, depending on the solution status returned from the MIP solver. If the current subproblem is proven infeasible, the value of α is increased in order to reduce the number of fixed variables in the next iteration (see line 16), and thus provide better diversification. Otherwise, if the time limit allowed for subproblem is exceeded without reaching a feasible solution or proving the subproblem infeasibility, the value of α is decreased. Decreasing the value of α, increases the number of fixed variables in the next iteration (see line 17), and thus reduces the size of the next subproblem. In the feasibility pump, the next reference integer solution is obtained by simply rounding the linear relaxation solution $\overline{x}$ of the modified problem LP$(\widetilde{x})$. However, if $near(\overline{x})$ is equal to some of the previous reference solutions, the solution process is caught in a cycle. In order to avoid this type of cycling, we determine the next reference solution as the one which is at the minimum distance from $near(\overline{x})$ (with respect to binary variables) and satisfies all constraints from the current subproblem Q (see line 18). In this way the convergence of the

variable neighbourhood diving algorithm is guaranteed (see [152]).

Algorithm 12: Single neighborhood diving for 0-1 MIP feasibility.

Function `SNDiving`(P);
Solve the LP relaxation of P to obtain an optimal LP basic solution $\overline{x}$;
Set $i = 0$; Set $\widetilde{x}^0 = [\overline{x}]$;
if $(\overline{x} = \widetilde{x}^0)$ **then return** $\widetilde{x}^0$;
Set $Q_0 = \emptyset$;
Set *proceed* = `true`; Set *timeLimit* for subproblems; Set value of α;
while *proceed* **do**
  Solve the $LP(P, \widetilde{x}^i)$ problem to obtain an optimal solution $\overline{x}$;
  if $(\lceil \delta(\widetilde{x}^i, \overline{x}) \rceil = 0)$ **then return** $\widetilde{x}^i$;
  $Q_{i+1} = (Q_i \mid \delta(\widetilde{x}^i, x) \geq \lceil \delta(\widetilde{x}^i, \overline{x}) \rceil)$;
  $\delta_j = \mid \tilde{x}_j - \overline{x}_j \mid$; index x_j so that $\delta_j \leq \delta_{j+1}$, $j = 1, \ldots, |\mathcal{B}| - 1$;
  $k = [\mid \{j \in \mathcal{B} : \widetilde{x}^i_j = \overline{x}_j\} \mid / \alpha]$; $J_k = \{1, \ldots, k\}$;
  $x' =$ `FindFirstFeasible`$(P(\widetilde{x}^i, J_k), timeLimit)$;
  if *feasible solution found* **then return** x';
  if $P(\widetilde{x}^i, J_k)$ is proven infeasible **then**
    $Q_{i+1} = (Q_{i+1} \mid \delta(J_k, \widetilde{x}^i, x) \geq 1)$; $P = (P \mid \delta(J_k, \widetilde{x}^i, x) \geq 1)$;
    $\alpha = 3\alpha/2$;
  else
    if *time limit for subproblem exceeded* **then** $\alpha = \max\{1, \alpha/2\}$;
  end
  $\widetilde{x}^{i+1} =$ `FindFirstFeasible`$(\text{MIP}(Q_{i+1}, [\overline{x}]), timeLimit)$;
  if $\text{MIP}(Q_{i+1}, [\overline{x}])$ *is proven infeasible* **then** Output message: "Problem P is proven infeasible"; **return**;
  $i = i + 1$;
end

1.6 Proximity heuristics

1.6.1 Ceiling heuristic

Saltzman et al. [199] in 1992, proposed a heuristic for general integer linear programming that was successfully applied for solving 0-1 MIP problems as well. The proposed heuristic is based on examination of so-called *"1-ceiling points"*, i.e., MIP feasible solutions located near to the boundary of the feasible region. More precisely, 1-ceiling point is a MIP feasible vector x such

that for each j at least one of vectors $x + e_j$ or $x - e_j$ is infeasible. The heuristic works in three phases:

- **Phase I**. In the first phase, the algorithm finds an optimal solution $\bar{x}$ of the LP relaxation of the problem, determines the set of constraints binding at that solution and the set of normalized extreme directions defining the cone originating at $\bar{x}$.

- **Phase II**. In the second phase, the algorithm chooses a hyperplane that is explored in a certain direction. As soon as some solution with an integer component value is encountered during the search, it is rounded to an integer solution that is not necessarily 1-ceiling point. The hyperplane to be explored is chosen as one along which the objective value changes as little as possible. More precisely, in a maximization problem, the objective function decreases as we move away from $\bar{x}$ along every extreme direction. So, let the rate of change of the objective function value per unit step taken away from $\bar{x}$ along direction d^k be ρ^k and E_i, be the set extreme directions emanating from $\bar{x}$ which lie on the i-th constraint hyperplane, i.e., $\sum_{j \in N} a_{ij} x_j \leq b_i$. Then the hyperplane i^* to be explored is chosen as $i^* = \arg\min_i \sum_{k \in E_i} \rho^k$. This hyperplane is explored in the direction $d = \sum_{k \in E_{i^*}} d^k$. Going in this direction through the chosen hyperplane as soon as a non-integer solution x with at least one integer component is met, it is rounded to the integer solution $\tilde{x}$ so that i^*-th constraint is satisfied. More precisely, let a_{i^*j} , $j \in N$ be the coefficients of i^*-th constraint, then components of solution $\tilde{x}$ are given as:

$$\tilde{x}_j = \begin{cases} \lfloor x_j \rfloor, & \text{if } a_{i^*j} > 0 \\ \lfloor x_j + 0.5 \rfloor, & \text{if } a_{i^*j} = 0 \\ \lceil x_j \rceil, & \text{if } a_{i^*j} < 0 \end{cases} \tag{1.10}$$

- **Phase III (Improvement phase)**. On a feasible integer solution found in the Phase II, Phase III procedures are launched in order to improve it, if possible, by altering either one or two of its components.

Each of these procedures, described hereafter, are capable to return a l-ceiling point (if any). These two procedures work in the following way:

- The first procedure, called STAYFEAS, attempts to improve a given feasible solution $\tilde{x}$ by altering just one of its components. In other words, STAYFEAS examines all integer solutions of the form $\tilde{x}' = \tilde{x} \pm e_j$, for all $j \in N$. The procedure returns the best feasible solution $\tilde{x}'$ (if any).
- The second procedure tries to improve a given feasible solution $\tilde{x}$ by simultaneously altering two of its components. That procedure consists of two steps. In the first step, just one component, e.g., $\tilde{x}_j$ is modified by either +1 or - 1 . If the obtained solution is feasible, the STAYFEAS procedure is applied on that solution, in an attempt to improve it further. On the other hand, if the obtained solution is not feasible, a second procedure named GAINFEAS is launched in order to get a feasible solution possibly better than $\tilde{x}$ by changing another component $k \neq j$ of the infeasible solution.

1.6.2 Relaxation Enforced Neighbourhood Search

The *Relaxation Enforced Neighborhood Search* heuristic (RENS) was proposed by Berthold in [20] as a new start heuristic for general MIPs working in the spirit of a large neighborhood search. RENS starts solving LP relaxation of the problem. After that all integer variables that received integer values in the solution of LP relaxation are fixed while on remaining variables, a large neighborhood search (LNS) is performed. The LNS is implemented solving resulting sub-MIP in which not only variables are fixed, but also all general integer variables with a fractional LP-value are rebounded to the nearest integers. If the sub-MIP is solved to optimality then the obtained solution is the best rounding of the fractional LP solution that any pure rounding heuristic can generate. Additionally, if the created sub-MIP is infeasible, then no rounding heuristic exists which is able to generate a feasible solution out of the fractional LP optimum.

1.6.3 Relaxation Induced Neighbourhood Search

The *Relaxation Induced Neighborhood Search* (RINS), is an improving heuristic proposed by Danna et al. [56]. The idea of RINS stems from the fact that often the incumbent solution of a MIP and the optimum of the LP-relaxation have many variables set to the same values. So, a partial solution, obtained by fixing these variables, may be likely extended to a complete integer solution with a good objective value. Therefore, RINS is focused on those variables whose values are different in the LP-relaxation and in the incumbent solution. In order to find appropriate values to such variables RINS explores the neighborhood structure defined by the incumbent solution $\tilde{x}$ and LP-relaxation solution $\bar{x}$ as:

$$\mathcal{RIN}(\tilde{x}, \bar{x}) = \{x | x_j = \tilde{x}_j \,\text{for}\, j \in \mathcal{I} \,\text{such that}\, \bar{x}_j = \tilde{x}_j; x \,\text{MIP feasible}\} \quad (1.11)$$

This neighborhood is called the relaxation induced neighborhood of $\tilde{x}$. In order, to efficiently explore this neighborhood, RINS solves the sub-MIP deduced from the original MIP, fixing the variables that have the same values in the incumbent and in the LP relaxation and adding the objective cut-off $cx \geq (1 + \theta)c\tilde{x}$, since we are interested in solutions that are better than the current incumbent solution. However, solving this sub-MIP might be so time consuming and thus it is preferable to solve sub-MIP approximatively, imposing the node limit or to call RINS only if a high enough percentage of variables can be fixed.

Since the continuous relaxation changes from one node in the branch-and-cut tree to the next, RINS may be invoked at every node or at some nodes in the tree in order to find high quality solutions of the initial MIP within the imposed time limit.

1.6.4 Proximity search heuristics

In 2013, Fischeti et al. [76] proposed a *Proximity search* heuristic (Algorithm 13) for 0-1 Convex Mixed Integer Programs. The purpose of the procedure is to refine a given feasible solution x^* of a problem. The basic procedure works iteratively, solving at each iteration a MIP derived from the original problem

by replacing the original objective function, by a "proximity objective" $\delta(x, x^*)$ (or $\delta(x, x^*) + \eta cx$ in order to favor high quality solutions) defined relative to the current iterate x^* and the objective function constraint $cx \geq (1 + \theta)cx^*$. The solution $\tilde{x}$ obtained by solving such defined MIP, is further improved by solving the initial problem fixing values of all binary variables to these in $\tilde{x}$. The procedure finishes its work when some of predefined stooping conditions is satisfied (e.g., max number of iterations, max CPU time allowed etc.). The steps of a Proximity search heuristic are given at Algorithm 13.

Algorithm 13: Proximity search heuristic for 0-1 MIP

Function PSH(x^*);
1 **repeat**
2 add the objective function constraint $cx \geq (1 + \theta)cx^*$ to the MIP ;
3 replace objective function by "proximity objective" $\delta(x, x^*)$ (or $\delta(x, x^*) + \eta cx$);
4 apply MIP solver in order to solve the new problem;
5 if MIP solver returns a solution $\tilde{x}$, refine it solving initial problem fixing values of all binary variables to these in $\tilde{x}$;
6 Let obtained solution be $\overline{x}$;
7 recenter search by setting $x^* = \overline{x}$ and/ or update value of θ;
until *Stopping criterion is satisfied*;
return x^*;

1.7 Advanced heuristics

1.7.1 Parametric tabu search

The *parametric tabu search* (PTS), proposed by Glover [97], is a general framework for building heuristics for solving general MIP problems. The main idea of PTS is solving series of linear programming problems deduced from the original MIP, incorporating branching inequalities as weighted terms in the objective function. This approach may be seen as an extension of a parametric branch-and-bound algorithm [91], that is accomplished replacing the branch-and-bound tree search memory by the adaptive memory framework of a tabu search. In that way more flexible strategies are introduced than those of Branch-and-Bound.

Following ideas described in [97], Sacchi et al. [198] in 2011, implemented

and tested the core parametric tabu search for solving 0-1 MIP problems. At each iteration, the core parametric tabu search solves LP problem deduced in the following way from the original MIP. Let x' be a so-called *trial solution* that is LP feasible. Relative to this solution, we may define sets:

$$N^1(x') = \{j \in \mathcal{I} | x'_j = 1\}$$

$$N^0(x') = \{j \in \mathcal{I} | x'_j = 0\}$$

that enable us to define so-called *goal conditions*:

$$(UP)\ x_j \geq 1, j \in N^1(x')$$

$$(DN)\ x_j \leq 0, j \in N^0(x')$$

Unlike Branch-and-Bound procedure, these conditions are not imposed implicitly, but indirectly incorporating them in the objective function. In that way the following LP problem is defined:

$$(LP(x', v^*)) \begin{cases} maximize\ cx + \sum_{j \in N^1(x')} c'_j(1 - x_j) + \sum_{j \in N^0(x')} c'_j x_j \\ s.t.\ Ax \leq b \\ 0 \leq x_j \leq U_j,\ j \in N \\ cx \geq (1 + \theta)v^* \end{cases} \tag{1.12}$$

where c'_j are positive parameters, v^* represents the best known solution value so far (initially $v^* = -\infty$) and θ is a small positive value.

After solving, the problem $LP(x', v^*)$ and obtaining its optimal solution x'', the next trial solution x', and therefore sets $N^+(x')$ and $N^-(x')$ are determined as the response to either the goal or integer infeasibility of the solution x''.

An optimal solution is called *goal infeasible* if there is some goal infeasible variable x''_j, i.e., variable:

$$\begin{cases} x''_j < 1 \text{ for } j \in N^1(x') \text{ or} \\ x''_j > 0 \text{ for } j \in N^0(x'). \end{cases} \tag{1.13}$$

The primary response for such an infeasibility consists of defining new goals in the opposite directions for a selected subset of goal infeasible variables $G_p \subset G = \{j \in N^1(x') \cup N^0(x') | x_j \text{ goal infeasible}\}$. More precisely, the primary response for $j \in G_p$ is defined as:

- if $x''_j < 1$ and $j \in N^1(x')$ then transfer j from $N^1(x')$ to $N^0(x')$ and set $x'_j = 0$,
- if $x''_j > 0$ and $j \in N^0(x')$ then transfer j from $N^0(x')$ to $N^1(x')$ and set $x'_j = 1$.

On the other hand, the secondary response, consists of freeing goal infeasible variables that belong to the set $G_s \subset G$. The elements of previously mentioned sets G_p and G_s are determined as g_p variables from the set G and g_s variables from the set $G - G_p$ with highest amounts of violation of imposed goal conditions.

An optimal solution x'' is called *integer infeasible* if there is some $j \in N^* = \mathcal{I} - N^1(x') - N^0(x')$ such that $x''_j \notin \{0, 1\}$. In order to respond to such an infeasibility, the subset N' of n' elements from the set N^* is chosen and each element from that set is added either to the set $N^1(x')$ or to the set $N^0(x')$, i.e., goal conditions are imposed for a certain number of variables. The choice of elements from the set N^* is based on the preference measure CP_j, that is calculated, relative to the up penalty $f_j^+ = \lceil x''_j \rceil - x''_j$ and the down penalty $f_j^- = x''_j - \lfloor x''_j \rfloor$, as

$$CP_j = (f_j^+ + f_j^-)/(f_j^+ - f_j^- + \omega)$$

where ω represents a small positive value. Once, n' elements from the set N^* are chosen as those with highest CP_j values, each of them is added either to set $N^1(x')$ or $N^0(x')$ depending on the values of f_j^+ and f_j^-. Namely, if $f_j^+ < f_j^-$ then j is added to $N^1(x')$, while otherwise it is added to $N^0(x')$. More precisely, the response for $j \in N'$ is defined as:

- if $f_j^+ < f_j^-$ then add j to $N^1(x')$ and set $x'_j = 1$
- if $f_j^+ \geq f_j^-$ then add j to $N^0(x')$ and set $x'_j = 0$

The steps of the core tabu search are presented at Algorithm 15. Initially, sets $N^1(x')$ and $N^0(x')$ are set to be empty and v^* is set to $-\infty$. After that at each iteration corresponding $LP(x', v^*)$ problem is treated. If it does not have a feasible solution, the core tabu search finishes its work, and the best found MIP solution (if any), regarding previous iterations is reported as the optimal one. Otherwise, the $LP(x', v^*)$ problem is solved. If its optimal solution x'' is MIP feasible, the best found solution value is updated and process is resumed solving new $LP(x', v^*)$ problem. Otherwise, the optimal solution x'' is either the goal infeasible or the integer infeasible. If the solution x'' is the goal infeasible, sets G_p and G_s are created choosing their elements taking into account tabu statuses of candidate elements. After that sets $N^1(x')$ and $N^0(x')$ are updated according to the responses associated with elements in G_p and G_s. Additionally, tabu tenures and aspiration values for the selected elements are updated as well. On the other hand, if the solution x'' is the integer infeasible, the set N' is created and sets $N^1(x')$ and $N^0(x')$ are updated according to the responses associated with elements in N'. Regardless of the encountered infeasibility, as soon as sets $N^1(x')$ and $N^0(x')$ are updated, the next iteration is invoked. The whole process is repeated until reaching the imposed stopping criterion (e.g., maximum number of iterations, maximum allowed CPU time etc.).

Algorithm 14: Core Tabu Search for 0-1 MIP

```
   Function CTS();
1  Choose x' ∈]0,1[^n;
2  v* = −∞;
3  repeat
4      if LP(x', v*) infeasible then break;
5      x'' ← optimal solution of LP(x', v*);
6      if x'' MIP feasible then
7          v* ← cx'';
8          x* ← x'';
9          continue;
       end
10     if x'' goal infeasible then
11         Create sets G_p and G_s;
12         Update sets N^1(x') and N^0(x');
13         Update tabu tenures and aspiration;
14         continue;
       end
15     if x'' integer infeasible then
16         Create set N' ;
17         Update sets N^1(x') and N^0(x');
18         continue;
       end
   until Stopping criterion is satisfied;
   return x*;
```

This core tabu search procedure may be extended to a more advanced procedure that includes intensification and diversification steps. For more details regarding the ideas for creating such one procedure as well as for description of the additional supporting strategies that may be used within it, we refer the reader to [97].

1.7.2 Metaheuristic Search with Inequalities and Target Objectives

Many adaptive memory and evolutionary metaheuristics for mixed integer programming include proposals for introducing inequalities and target objec-

tives to guide the search toward an optimal (or near-optimal) solution. These guidance approaches consist of fixing subsets of variables at particular values and using linear programming to generate trial solutions whose variables are induced to receive integer values. Such one approaches may be used in both intensification and diversification phases of a solution process. In 2010, Glover and Hanafi [99, 100] enhanced these approaches by introducing new inequalities that dominate those previously proposed and new target objectives that underlie the creation of both inequalities and trial solutions. More precisely, they proposed use of partial vectors and more general target objectives within inequalities in target solution strategies. Actually, they proposed procedures for generating target objectives and solutions by exploiting the proximity in the original space or the projected space. Additionally, they introduced more advanced approaches for generating the target objective based on exploiting the mutually reinforcing notions of reaction and resistance.

The proximity procedure (Algorithm 16) for solving pure 0-1 MIP problems ($\mathcal{I} = N$), proposed in [99, 100], at each iteration solves the linear program defined as:

$$(LP(x', c', v^*)) \begin{cases} minimize\ \delta(c', x', x) = \sum_{j=1}^{n} c'_j(x_j(1 - x'_j) + x'_j(1 - x_j)) \\ s.t.\ Ax \leq b \\ 0 \leq x_j \leq U_j,\ j \in N \\ cx \geq (1 + \theta)v^* \end{cases} \tag{1.14}$$

where $\delta(c', x', x)$ is the target objective and c' is an integer vector.

Initially, the vector c is used as the vector c', while the target solution x' is obtained setting its components to 0 (i.e., in the first iteration the initial LP problem is solved). After that, for each next iteration, a vector c' and a new target solution x' are deduced from the optimal solution x'' of the last solved $LP(x', c', v^*)$ problem. The new target solution x' is derived from x'' simply by setting $x'_j = near(x''_j), j \in N$. The resulting vector x' of the nearest integer neighbors is unlikely to be 0-1 MIP feasible. If the solution x' is 0-1 MIP feasible, it is stored as a new best solution x^*, the objective function constraint $cx \geq (1 + \theta)v^*$ is updated, and target objective $\delta(c', x', x)$ is set to the initial objective cx (i.e., in the next iteration the original LP with the

updated objective function constraint will be solved). On the other hand, if the solution x' is 0-1 MIP infeasible, the vector c' is generated, so that the solution x'' of the next generated problem $LP(x', c', v^*)$ will become closer to satisfying integer feasibility. The generation is accomplished by the following procedure (see [99] for more details):

Algorithm 15: Procedure for generating vector c'

Function `Generate_vector`$(BaseCost, x', x'')$;
Choose $\lambda_0 \in [0.1, 0.4]$;
for $j \in N$ **do**
  if $x''_j \notin]\lambda_0, 1-\lambda_0]$ **then**
    $c'_j = 1 + BaseCost(1 - 2x'_j)(0.5 - x''_j)/(0.5\lambda_0)$;
  else
    $c'_j = 1 + BaseCost(x'_j - x''_j)/\lambda_0$;
  end
end
return c';

The rationale for using such procedure is that targeting of $x_j = x'_j$ for variables whose values x''_j already equal or almost equal x'_j does not have great impact on the solution of the new (updated) $LP(x', c', v^*)$, in the sense that such a targeting does not yield the solution that differ substantially from the solution to the previous $LP(x', c', v^*)$ problem. Therefore, it is more beneficial if targeting occurs by emphasizing the variables x_j whose x''_j values differ from their integer neighbours x'_j by a greater amount. Note that according to [99] the suggested value for parameter $BaseCost$, of procedure for generating vector c', is 20.

As a stopping criterion the proximity procedure may use the total number of iterations allowed or the number of iterations since finding the last feasible

integer solution etc.

Algorithm 16: Proximity procedure for 0-1 MIP

```
   Function PS(BaseCost);
1  c' = c;
2  x' = 0;
3  v* = −∞;
4  repeat
5      if LP(x', c', v*) infeasible then break;
6      x'' ← optimal solution of LP(x', c', v*);
7      if x'' integer feasible then
8          x* ← x''; //update the best solution
9          v* = cx''; //update the objective function constraint
10         x' = 0, c' = c;
       else
11         x'_j = near(x''_j), for j ∈ N; //Construct the target solution x' derived from
           x''
12         c' ← Generate_vector (BaseCost, x', x'');
       end
   until Stopping criterion is satisfied;
   return x*;
```

The described proximity procedure, may be easily enhanced by "updating the problem inequalities" (adding and dropping constraints) in the way described in [99]. Further, in order to avoid big difference between the components of two vectors c', used in two consecutive iterations, it is preferred not to change all the components of c' each time a new target objective is produced, but to change only a subset consisting of k of these components. For example, a reasonable default value for k is given by $k = 5$. Alternatively, the procedure may begin with $k = n$ and gradually reduce k to its default value or to allow it to oscillate around a preferred value. The k components of c' that will be changed may be chosen as those k having the k largest c'_j values in the new target objective.

The merit of a used target objective $\delta(c', x', x)$ may be expressed in terms of "reaction" and "resistance". The term reaction refers to the change in the value of a variable as a result of creating a target objective $\delta(c', x', x)$ and solving the resulting problem $LP(x', c', v^*)$. The term resistance refers to

the degree to which a variable fails to react to a non-zero c'_j coefficient by receiving a fractional value rather than being driven to 0 or 1. Hence, the proximity procedure may be enhanced introducing advanced approaches for generating the target objective based on exploiting the mutually reinforcing notions of reaction and resistance as proposed in [100].

1.7.3 OCTANE heuristic

In 2001, Balas et al. [11] proposed the OCTAhedral Neighbourhood Enumeration (OCTANE) heuristic for pure 0-1 programs. The fundamentals of OCTANE rely on the one-to-one correspondence between 0-1 points in n–dimensional space and the facets of the n–dimensional octahedron. More precisely, let a cube be given as $K = \{x \in \mathbb{R}^n : -1/2 \leq x \leq 1/2\}$ and a regular octagon K^* circumscribing this n-dimensional cube given by $K^* = \{x \in \mathbb{R}^n : \sigma x \leq n/2,\ \sigma \in \{\pm 1\}^n\}$. Hence, it follows that every facet σ of the octahedron K^* contains exactly one vertex τ of the hypercube K, namely the one with $\tau_j = 1/2$ if $\sigma_j = 1$ and $\tau_j = -1/2$ if $\sigma_j = -1$, i.e. $\tau_j = \sigma_j/2$.

Since both sets have the same cardinality, every vertex of the hypercube is as well contained in exactly one facet of the octahedron. Note that this correspondence is kept even if we translate K and K^* for the same vector. The basic idea of OCTANE is that finding facets that are near an LP feasible point, x^0 is equivalent to finding integer feasible points that are near x^0 and therefore potentially LP-feasible themselves. Furthermore, if we choose an optimal solution of the LP-relaxation as an initial point $\overline{x}$, the integer feasible solutions potentially will be of high quality in terms of the objective function. So, OCTANE starts by solving the LP-relaxation of the 0-1 program, then from $\overline{x}$, a fractional solution of the LP-relaxation of the 0-1 program, it computes the first k facets of an octahedron that are intersected by the half line originating at $\overline{x}$ and having a selected direction d. In this way OCTANE yields k potential 0-1 points of the original 0–1 programming problem. The

steps of OCTANE are given at Algorithm 17.

Algorithm 17: OCTANE for pure 0-1 MIP

Function `OCTANE` (P);

1 Solve the LP relaxation of P to obtain an optimal LP basic solution $\overline{x}$;
2 Choose a direction vector d and consider the half-line $L = \overline{x} + \lambda d$, $\lambda \geq 0$;
3 Compute the first k facets of an octahedron that are intersected by the half line L;
4 Transform k reached facets to 0-1 points;
5 Check the integer feasible points;
6 Report the best found feasible solution (if any);

A Ray Shooting Algorithm. In order to find the first k facets of an octahedron that are intersected by the half line originating at $\overline{x}$ and having a selected direction d a ray shooting algorithm is applied. Before starting to describe the Ray Shooting Algorithm terms reachable facet and first reachable facet will be defined.

Definition. A facet $\sigma \in \{\pm 1\}^n$ is called reachable, with respect to the given ray $r(\lambda) = x + \lambda d$ if there exists a $\lambda > 0$ for which $\sigma r(\lambda) = n/2$. A facet is called *first-reachable*, if the parameter λ is minimal among all of reachable facets.

The ray shooting algorithm may be described in the following way. Firstly, some facet is generated, which is definitely reachable. After that a first-reachable facet is determined by changing components of this facet. Finally, one performs a reverse search in order to find $k-1$ further facets. The first step, namely finding any reachable facet, is trivial. (After some transformation the reachable facet may be determined as $\sigma = e$ see [19]). After that the first reachable facet is deduced according to the following theorem:

Theorem 1.7.1 *If a facet σ is reachable, but not first-reachable, there exists an $i \in N$ for which the facet $\sigma \diamond i$ defined by: $(\sigma \diamond i)_j := -\sigma_j$ if $j = i$ and σ_j otherwise; is a facet which is hit before σ. Then i is called a decreasing flip. Then starting with $\sigma = e$ and iteratively flipping its components, if they yield a decreasing flip, is an algorithm which has a first-reachable facet, i.e. σ^* as output and can be implemented with a running time of $O(nlog(n))$.*

Thanks to this theorem, we know how to get a facet which is first-reachable.

However there could be more than one first-reachable facet. In practice this is rather the rule, if one chooses ray directions not randomly, but following some certain geometric structure.

Reverse search. Reverse search is used to determine the k first reachable facets. The idea of reverse search is to build up an arborescence rooted at σ^* which vertices are all reachable facets and after that to determine the k first reachable facets. Therefore, in order to form the arborescence we need to determine a unique predecessor $pred(\sigma)$ for each reachable facet σ, except for one first-reachable facet which will be the root-node of the arborescence.

Definition. An index $i \in N$ is called a

- decreasing + to - flip for σ, if $\sigma_i = +1$ and $\sigma \diamond i$ is a facet hit before σ
- nonincreasing - to + flip for σ, if $\sigma_i = -1$ and $\sigma \diamond i$ is a facet hit by r but not after σ

If there exists at least one decreasing + to - flip for σ, then, $pred(\sigma) := \sigma \diamond i$ where i corresponds to minimal index of all decreasing flips. On the other hand, if there is no decreasing + to - flip for σ, but at least one nonincreasing - to + flip, then, $pred(\sigma) := \sigma \diamond i$ where i corresponds to maximal index of all nonincreasing flips. Otherwise, $pred(\sigma)$ stays undefined. One can prove that there is exactly one facet σ^* for which $pred(\sigma)$ is undefined and obviously this is a first-reachable facet.

Then, as it shown in Balas et al. [11], the arcs of the arborescence are defined as $(pred(\sigma), \sigma)$ with associated weight that equals to distance between points σ and $pred(\sigma)$. Additionally, Balas et al. [11] proved the following statement.

Theorem 1.7.2 *There is an $O(knlog(k))$ algorithm which finds k vertices of the arborescence with minimum distance to σ^*. Moreover, these k vertices correspond to the k facets of K^* first hit by $r(\lambda)$.*

Selection of the Ray Direction. The ray direction may be chosen in some of the following ways:

- **The objective ray**: it is created choosing the direction opposite to the direction of objective function, i.e., $d = -c$. Clearly such ray leads into

the inner of the polyhedron and therefore promises to produce feasible solutions.

- **The difference ray**: The difference between the optimum of the LP-relaxation at the root-node of the branch-and-bound tree and the current LP–optimum in some node of the branch–and–bound tree shows a part of the development of each variable from the value in an optimal LP-solution to the one in an integer feasible solution. Therefore, this difference vector seems to be a ray direction with a promising geometric interpretation.

- **The average ray**: The optimal basis for the LP-relaxation at the current node defines a cone C which extreme rays are defined by edges of the current LP-polyhedron. Obviously, the average of the normalized extreme rays of C points into the inner of the LP polyhedron and therefore, hopefully into the direction of some feasible solutions.

- **The average weighted slack ray**: This ray is obtained from the average ray by additionally assigning weights to the extreme rays. Every extreme ray corresponding to a non-basic slack variable with positive reduced costs gets the inverse of the reduced costs as weights, while all others weights are assigned to 0.

- **The average normal ray [19]:** The LP-optimum is a vertex of the LP polyhedron and therefore, at least n of the linear and bounding constraints are fulfilled with equality. Therefore the normalized (inner) normal of a hyperplanes corresponding to some linear constraint gives a direction where all points are feasible for this constraint. So, the average of all these normals, probably, represent a direction of finding feasible points.

1.7.4 Star Path with directional rounding

The introduction of Star Paths with directional rounding for 0–1 Mixed Integer Program as a supporting strategy for Scatter Search in [93] established basic properties of directional rounding and provided efficient methods for exploiting

them. The most important of these properties is the existence of a plane (which can be required to be a valid cutting plane for MIP) which contains a point that can be directionally rounded to yield an optimal solution and which, in addition, contains a convex subregion all of whose points directionally round to give this optimal solution. Several alternatives are given for creating such a plane as well as a procedure to explore it using principles of Scatter Search. That work also shows that the set of all 0–1 solutions obtained by directionally rounding points of a given line (the so–called star path) contains a finite number of different 0–1 solutions and provides a method to generate these solutions efficiently. Glover and Laguna [101] elaborated these ideas and extended them to General Mixed Integer Programs by means of a more general definition of directional rounding. In Chapter 4 of this thesis, we provide thorough description of Star Paths with directional rounding for 0–1 MIP and propose Convergent algorithms of Scatter Search and Star Paths with Directional Rounding for 0-1 MIP along with the proof of their finite convergence.

Building on above ideas, Glover et al. [106] proposed a procedure that combines Scatter Search and the Star Path generation method as a basis for finding a diverse set of feasible solutions for 0–1 Mixed Integer Problems, proposing a 3–phase algorithm which works as follows. The first step generates a diverse set of 0–1 solutions using a dichotomy generator. After that each solution generated in a previous phase is used to produce two center points on an LP polyhedron which are further combined to produce sub-centers. All centers and sub–centers are combined in the last phase to produce Star–Paths. As output the algorithm gives the set of all 0–1 feasible solutions encountered during its execution, and constitutes the required diverse set. The computational efficiency of the approach was demonstrated by tests carried out on some instances from MIPLIB.

1.8 Concluding remarks

This chapter provides a survey of heuristics based on mathematical programming for 0-1 mixed integer programs. For most of considered heuristics we

not only describe their main ideas and their work but also provide their pseudo-codes written in the same fashion. Most of these heuristics are already embedded in many commercial solvers such as CPLEX, GUROBI, GAMS, XPRESS etc., but there is still a need for new heuristics that will additionally accelerate exact methods. Therefore the potential future research directions may be developing new advanced heuristics combining the ideas of existing not only exact but also heuristic approaches. Some steps in that direction are already done. For example, variable neighborhood search decomposition heuristic is combined with iterative linear programming heuristic. Further, some of the described heuristics may be seen as frameworks for creating heuristics, but they are not exploited (or seldom ever) in that way. Thus, the future research direction may also include implementing and testing new 0-1 MIP heuristics that stems from these frameworks with and without exploiting ideas of the existing heuristics. Some steps in this direction have been already done, and the results will be presented in Chapters 3 and 4.

Chapter 2

Variable Neighborhood Search

An optimization problem may be stated in a general form as:

$$\min\{f(x)|x \in X \subseteq \mathcal{S}\} \tag{2.1}$$

where $\mathcal{S}, X, x$ and f respectively denote the solution space and the feasible set, a feasible solution and a real-valued objective function. Depending on the set $\mathcal{S}$ we distinguish combinatorial optimization problem (set $\mathcal{S}$ is finite but extremely large) and continuous optimization problems ($\mathcal{S} = \mathbb{R}^n$). Let $\mathcal{N}(x)$ denote a neighborhood structure of a given solution x defined relatively to a given metric (or quasi-metric) function δ introduced in the solution space $\mathcal{S}$ as:

$$\mathcal{N}(x) = \{y \in X|\delta(x, y) \leq \alpha\}$$

where α is given positive number. Then, a solution $x^* \in \mathcal{N}(x)$ is a local minimum, relatively to neighborhood $\mathcal{N}(x^*)$, for problem (2.1) if $f(x^*) \leq f(x),\ \forall x \in \mathcal{N}(x^*)$. On the other hand, a solution $x^* \in X$ is an optimal solution (global optimum) for problem (2.1) if $f(x^*) \leq f(x),\ \forall x \in X$.

Many practical instances of problems of form (2.1), cannot be solved exactly (i.e., providing an optimal solution along with the proof of its optimality) in reasonable time. It is well-known that many problems of form (2.1) are NP-hard, i.e., no algorithm with a number of steps polynomial in the size of the instances is known for solving any of them and that if one were found it would be a solution for all. Therefore, there is a need for heuristics able to quickly produce an approximate solution of high quality, or sometimes an optimal solution but without proof of its optimality.

Variable neighborhood search (VNS) is a metaheuristic proposed by Mladenović and Hansen in 1997 [167]. It represents a flexible framework for building heuristics for approximately solving combinatorial and non-linear continuous

optimization problems. VNS changes systematically neighborhood structures during the search for an optimal (or near-optimal) solution. The changing of neighborhood structures is based on the following observations: (*i*) A local optimum relatively to one neighborhood structure is not necessarily a local optimal for another neighborhood structure; (*ii*) A global optimum is a local optimum with respect to all neighborhood structures; (*iii*) Empirical evidence shows that for many problems all local optima are relatively close to each other. The first property is exploited by increasingly using complex moves in order to find local optima with respect to all neighborhood structures used. The second property suggests using several neighborhoods, if local optima found are of poor quality. Finally, the third property suggests exploitation of the vicinity of the current incumbent solution.

A variable neighborhood search heuristic (Algorithm 28) includes an *improvement phase* used to possible improve a given solution and one so-called *shaking phase* used to hopefully resolve local minima traps. The improvement phase and the shaking procedure, together with neighborhood change step are executed alternately until fulfilling a predefined stopping criterion. As stopping criterion, most often, is used maximum CPU time allowed to be consumed by VNS heuristic. VNS heuristics have been successfully applied for solving many optimization problems (see e.g., [126]).

Algorithm 18: Basic steps of Variable Neighborhood Search.

```
Function VNS()
repeat
1   Shaking_procedure;
2   Improvement_procedure;
3   Neighborhood_change;
until stopping criterion is fulfilled;
```

The rest of the chapter is organized as follows. In Section 2.1 we present the most common improvement procedures used within a VNS heuristic such as local search and variable neighborhood descent variants, while in Section 2.2 we describe the simple shaking procedure that is usually used within a VNS heuristic. Section 2.3 is dedicated to VNS variants, while Section 2.4 contains the brief description of VNS heuristics developed by the author

for solving several optimization problems along with their comparison with the state-of-the-art heuristics for the considered problems. The optimization problems for which VNS based heuristics have been developed are: Traveling salesman problem with time windows, Attractive traveling salesman problem, Traveling salesman problem with draft limits, Swap-body vehicle routing problem, Uncapacitated r-allocation p-hub median problem, Minimum sum-of-squares clustering on networks, Periodic maintenance problem and Unit commitment problem. Note that the exhaustive description of the proposed VNS methods as well as the detailed computational results for each considered problem may be found in the journal articles [36, 169, 170, 210, 213] as well as in the reports [209, 211, 212]. Finally, in Section 2.5, we draw some conclusions and present possible future research lines.

2.1 Improvement procedures used within VNS

2.1.1 Local search

A local search heuristic is based on the exploration of a neighborhood structure $\mathcal{N}(x)$ of a current incumbent solution x at each iteration. Starting from an initial solution x, at each iteration it selects a better solution than x' (if any) from the predefined neighborhood structure $\mathcal{N}(x)$ and sets it to be the new incumbent solution x. A local search heuristic finishes its work reaching an incumbent solution x such that it is already the local optimum with respect to its neighborhood structure $\mathcal{N}(x)$. The most common search strategies used within a local search heuristic are the *first improvement* search strategy (as soon as an improving solution in a neighborhood structure $\mathcal{N}(x)$ is detected it is set to be the new incumbent solution) and the *best improvement* search strategy (the best among all improving solutions in $\mathcal{N}(x)$ (if any) is set to be the new incumbent solution). The steps of a local search heuristic using the first improvement strategy and a local search heuristic using the best improvement strategy are given in algorithms 19 and 20, respectively.

Algorithm 19: Local search using the first improvement search strategy.

```
Function LS_FI(x, 𝒩)
repeat
    Let 𝒩(x) = {x^1, ..., x^p};
    i ← 0;
    x' ← x;
    repeat
        i ← i + 1;
        if f(x^i) < f(x) then
            x ← x^i
            break;
        end
    until i = p;
until f(x') ≤ f(x);
return x';
```

Algorithm 20: Local search using the best improvement search strategy.

```
Function LS_BI(x, 𝒩)
repeat
    x' ← x;
    x ← argmin_{y∈𝒩(x')} f(y);
until f(x') ≤ f(x);
return x';
```

2.1.2 Variable neighborhood descent procedures

The Variable Neighborhood Descent (VND) procedures exploit the fact that the solution which is a local optimum with respect to several neighborhood structures is more likely to be a global optimum than the solution generated as a local optimum for just one neighborhood structure. More precisely a VND procedure explores several neighborhood structures either in a sequential or nested fashion in order to possible improve a given solution. As a search strategy they may use either the first improvement strategy or the best improvement search strategy.

Sequential variable neighborhood descent procedures

The standard sequential VND (seqVND) procedure works in the following way. Several neighborhood structures are firstly ordered in a list and after that examined one after another respecting the established order. Let $\mathcal{N} = \{\mathcal{N}_1, \ldots, \mathcal{N}_{k_{max}}\}$ be set of operators such that each operator $\mathcal{N}_k$, $1 \leq k \leq k_{max}$ maps a given solution x to a predefined neighborhood structure $\mathcal{N}_k(x)$, (i.e., $\mathcal{N}_k : X \rightarrow \mathcal{P}(X)$, where $\mathcal{P}(X)$ denotes the power set of the set X). Note that the order of operators in the set $\mathcal{N}$ also will define the order of examining neighborhood structures of a given solution x. Starting from a given solution x, the standard sequential VND procedure iteratively explores its neighborhood structures defined, by the set $\mathcal{N}$, one after another according to the established order. As soon as an improvement of the incumbent solution in some neighborhood structure occurs, the standard sequential VND procedure resumes search in the next neighborhood structure (according to the defined order) of the new incumbent solution. The whole process is stopped if the current incumbent solution can not be improved with respect to any of k_{max} neighborhood structures. The steps of the sequential VND using the best improvement search strategy are given in Algorithm 22. Besides of this standard sequential VND procedure, the two sequential VND procedures have been proposed using the different rules for selecting the next neighborhood structure to be explored if an improvement of current incumbent solution occurs:

- **pipe VND** : if an improvement of the current solution occurs in some neighborhood, the exploration is continued in that neighborhood. The steps of pipe VND using the best improvement search strategy may be deduced from the Algorithm 22, replacing the neighborhood change procedure `Neighborhood_change_sequential(`x`, `x'`, `k`)` given in Algorithm 21 by the procedure `Neighborhood_change_pipe(`x`, `x'`, `k`)` given in Algorithm 23. Note that the pipe VND sometimes is referred as the token ring search (see e.g., [59, 158]).

- **cyclic VND** : regardless there is an improvement with the respect to some neighborhood or not, the search is continued in the next

neighborhood structure in the list. Therefore, the steps of cyclic VND using the best improvement search strategy may be deduced from the Algorithm 22, replacing the neighborhood change procedure `Neighborhood_change_sequential` given in Algorithm 21 by the procedure `Neighborhood_change_cyclic(`x`, `x'`, `k`)` given in Algorithm 24. The approach presented in [103] could be considered as a cyclic VND except that it has cycles within cycles and uses additional evaluation criteria along the way.

Note that each of these VND algorithms may also use the first improvement search strategy to explore neighborhood structures. If the first improvement strategy is used, the steps of each VND variant are the same as the steps of corresponding VND variant that uses best improvement strategy but with additional assumption that call $argmin_{y \in \mathcal{N}_k(x)} f(y)$ returns the first encountered solution y in the neighborhood $\mathcal{N}_k(x)$ such that$f(y) < f(x)$.

Algorithm 21: Neighborhood change step for sequential VND.

Procedure `Neighborhood_change_sequential`(x, x', k)
if $f(x') < f(x)$ **then**
  $x \leftarrow x'$;
  $k \leftarrow 1$;
else
  $k \leftarrow k + 1$;
end

Nested variable neighborhood descent

A nested variable neighborhood descent procedure [133] explores a large neighborhood structure obtained as a composition of several neighborhoods. More precisely, let $\mathcal{N} = \{\mathcal{N}_1, \ldots \mathcal{N}_{k_{max}}\}$ again be set of operators such that each operator $\mathcal{N}_k$, $1 \leq k \leq k_{max}$ maps a given solution x to a predefined neighborhood structure $\mathcal{N}_k(x)$. Then, the neighborhood explored within a nested variable neighborhood procedure is defined by operator $\mathcal{N}^* = \mathcal{N}_1 \circ \mathcal{N}_2 \circ \cdots \circ \mathcal{N}_{k_{max}}$. Obviously, the cardinality of a neighborhood structure $\mathcal{N}^*(x)$ of some solution x equals to product of cardinalities of nested (composed)

Algorithm 22: Steps of sequential VND using the best improvement search strategy.

Function `SeqVND`(x, k_{max}, $\mathcal{N}$)
repeat
  $stop \leftarrow false$;
  $k \leftarrow 1$;
  $x' \leftarrow x$;
  repeat
    $x'' \leftarrow \text{argmin}_{y \in \mathcal{N}_k(x)} f(y)$;
    `Neighboorhood_change_sequential`(x, x'', k);
  until $k = k_{max}$;
  if $f(x') \leq f(x)$ **then**
    $stop \leftarrow true$;
  end
until *stop = true;*
retourner x';

Algorithm 23: Neighborhood change step for pipe VND.

Procedure `Neighborhood_change_pipe`(x, x', k)
if $f(x') < f(x)$ **then**
  $x \leftarrow x'$;
else
  $k \leftarrow k + 1$;
end

Algorithm 24: Neighborhood change step for cyclic VND.

Procedure `Neighborhood_change_cyclic`(x, x', k)
$k \leftarrow k + 1$;
if $f(x') < f(x)$ **then**
  $x \leftarrow x'$;
end

neighborhoods, i.e.,

$$|\mathcal{N}^*(x)| = \prod_{k=1}^{k_{max}} |\mathcal{N}_k(x)|$$

Such cardinality obviously increases chances to find an improvement in that nested neighborhood. The steps of nested VND using the best improvement

search strategy are given in Algorithm 25. The neighborhood $\mathcal{N}^*(x)$ may be also explored using the first improvement search strategy, following steps of local search procedure using the first improvement strategy given in Algorithm 19.

Algorithm 25: Steps of nested VND using the best improvement search strategy.

Function `Nested_VND`(x, k_{max}, $\mathcal{N}$)
$\mathcal{N}^* = \mathcal{N}_1 \circ \mathcal{N}_2 \circ \cdots \circ \mathcal{N}_{k_{max}}$
repeat
  $x' \leftarrow x$;
  $x \leftarrow argmin_{y \in \mathcal{N}^*(x')} f(y)$;
until $f(x') \leq f(x)$;
return x';

Mixed variable neighborhood descent

Mixed variable neighborhood descent [133] combines ideas of sequential and nested variable neighborhood descent. Namely, it uses a set of operators $\mathcal{N} = \{\mathcal{N}_1, \ldots \mathcal{N}_{k_{max}}\}$ to define a nested neighborhood and after that on each element in this nested neighborhood applies sequential variable neighborhood descent variant defined by a set of operators $\mathcal{N}' = \{\mathcal{N}'_1, \ldots \mathcal{N}'_{\ell_{max}}\}, \mathcal{N}' \cap \mathcal{N} = \emptyset$. The cardinality of the set explored in one iteration of mixed VND is bounded by $\sum_{\ell=1}^{\ell_{max}} |\mathcal{N}'_\ell(x)| \times \prod_{k=1}^{k_{max}} |\mathcal{N}_k(x)|, x \in X$. Note that ideas similar to those of mixed VND have been exploited in the filter&fan and the ejection chains methods, see [143, 187] and the references therein. In Algorithm 26 we give steps of mixed VND which uses the best improvement search strategy within the nested neighborhood and standard sequential VND presented in Algorithm 22 which also uses the best improvement search strategy. Note that mixed VND may be implemented using the first improvement search strategy either within nested neighborhood or sequential VND as well as using pipe or cyclic VND to explore the nested neighborhood.

Algorithm 26: Steps of mixed VND using the best improvement search strategy.

Function `Mixed_VND`(x, k_{max},ℓ_{max}, $\mathcal{N}$, $\mathcal{N}'$)
$\mathcal{N}^* = \mathcal{N}_1 \circ \mathcal{N}_2 \circ \cdots \circ \mathcal{N}_{k_{max}}$
$x' \leftarrow x$;
repeat
  $stop = true$;
  $x \leftarrow x'$;
  for *each* $y \in \mathcal{N}^*(x)$ **do**
    $x'' \leftarrow$ `SeqVND`(y, ℓ_{max}, $\mathcal{N}'$) ;
    if $f(x'') < f(x')$ **then**
      $stop = false$
      $x' \leftarrow x''$;
    end
  end
until $stop = true$;
return x';

2.2 Shaking procedure

The aim of a shaking procedure used within a VNS heuristic is to hopefully resolve local minima traps. The simple shaking procedure consists of selecting a random solution from the k^{th} neighborhood structure. For some problem instances, a completely random jump in the k^{th} neighborhood is too diversified. Hence, sometimes is preferable to do so-called intensified shaking checking taking into account how sensitive is the objective function to small change (shaking) of the solution. However, for the sake of simplicity we will assume that each VNS variant presented hereafter uses a simple shaking procedure based on selecting a random solution from the k^{th} neighborhood structure (see Algorithm 27).

Algorithm 27: Shaking procedure

Function `Shake`$(x, k, \mathcal{N})$;
choose $x' \in \mathcal{N}_k(x)$ at random;
return x'

2.3 Variable neighborhood search variants

The basic VNS variant executes alternately a simple local search procedure (one of two presented in Algorithms 19 and 20) and a shaking procedure presented in Algorithm 27 together with neighborhood change step (presented in Algorithm 21) until fulfilling a predefined stopping criterion. As a stopping criterion, most often, is used maximum CPU time allowed to be consumed by Basic VNS. Its steps are given in Algorithm 28.

Algorithm 28: Basic Variable Neighborhood Search.

Function `Basic_VNS`(x, k_{max}, $\mathcal{N}$,$\mathcal{N}'$)
repeat
  $k \leftarrow 1$;
  while $k \leq k_{max}$ **do**
    $x' \leftarrow$ `Shake`$(x, k, \mathcal{N})$;
    $x'' \leftarrow$ `Local_search`$(x', \mathcal{N}')$;
    `Neighborhood_change_sequential`(x, x'', k);
  end
until *stopping condition is fulfilled;*
return x;

From this basic VNS scheme several other VNS approaches have been derived. The simplest one is so-called *Reduced VNS*, which employees a shaking procedure and a neighborhood change step procedure while the improvement phase is discarded. The steps of *Reduced VNS* that uses sequential neighborhood change function are given in Algorithm 29.

Another variant is so-called *General VNS* which as an improvement procedure uses some of VND procedure presented above unlike to Basic VNS which uses just a simple local search. Note that it is not obligatory that neighborhood structures used within the shaking procedure and a VND procedure are the same although it is more desirable. The steps of a General VNS are given in Algorithm 30. In the algorithm the statement of the form `VND`(x', ℓ_{max}, $\mathcal{N}'$) means that a certain VND variant presented above is executed.

Skewed VNS variants are those that in the neighborhood change step accept as new incumbent solutions not only improving solutions but also those

Algorithm 29: Reduced Variable Neighborhood Search.

Function `Reduced_VNS`(x, k_{max}, $\mathcal{N}$)
repeat
  $k \leftarrow 1$;
  while $k \leq k_{max}$ **do**
    $x' \leftarrow$ `Shake`$(x, k, \mathcal{N})$;
    `Neighborhood_change_sequential`(x, x', k);
  end
until *stopping condition is fulfilled;*
return x;

Algorithm 30: General Variable Neighborhood Search.

Function `General_VNS`(x, k_{max}, ℓ_{max}, $\mathcal{N}$,$\mathcal{N}'$)
repeat
  $k \leftarrow 1$;
  while $k \leq k_{max}$ **do**
    $x' \leftarrow$ `Shake`$(x, k, \mathcal{N})$;
    $x'' \leftarrow$ `VND`(x', ℓ_{max}, $\mathcal{N}'$);
    `Neighborhood_change_sequential`(x, x'', k);
  end
until *stopping condition is fulfilled;*
return x;

worse than the current incumbent solution. The purpose of such neighborhood change step is to allow exploration of valleys far from the incumbent solution. Therefore, in the so-called *skewed neighborhood change step*, a trial solution is evaluated taking into account not only the objective values of the trial and the incumbent solution but also the distance between them. The evaluation function used in the skewed neighborhood change step may be stated as:

$$g(x, y) = f(x) - f(y) - \alpha d(x, y)$$

where α represents positive parameter while $d(x, y)$ represents the distance between solutions x and y. The *skewed neighborhood change step* using this function is given in Algorithm 31.

One generalization of VNS is *nested VNS*. It may be also seen as a

Algorithm 31: Neighborhood change step for skewed VNS.

Procedure `Skewed_Neighborhood_change`(x, x', k, α)
if $f(x') - f(x) < \alpha d(x', x)$ **then**
  $x \leftarrow x'$;
  $k \leftarrow 1$;
else
  $k \leftarrow k + 1$;
end

generalization of mixed VND that consists of applying a VNS variant on each point of a predefined neighborhood structure instead of applying a VND variant. In algorithm 32 we give steps of nested VNS which uses best improvement search strategy to explore the given neighborhood structure (although the first improvement search strategy may be used as well). For performance of *nested VNS* it is crucial that the VNS applied at each point of the predefined neighborhood is very fast. For that reason it is desirable to use small CPU time limit as a stooping criterion for this VNS.

Algorithm 32: Steps of nested VNS using the best improvement search strategy.

Function `Nested_VNS`(x, k_{max},ℓ_{max}, ℓ'_{max} $\mathcal{N}$, $\mathcal{N}'$, $\mathcal{N}''$)
$\mathcal{N}^* = \mathcal{N}_1 \circ \mathcal{N}_2 \circ \cdots \circ \mathcal{N}_{k_{max}}$;
$x' \leftarrow x$;
repeat
  $stop = true$;
  $x \leftarrow x'$;
  for *each* $y \in \mathcal{N}^*(x)$ **do**
    $x'' \leftarrow$ `VNS`(y, ℓ_{max}, ℓ'_{max}, $\mathcal{N}'$, $\mathcal{N}''$) ;
    if $f(x'') < f(x')$ **then**
      $stop = false$;
      $x' \leftarrow x''$;
    end
  end
until $stop = true$;
return x';

The previously described nested VNS may be used as the improvement

procedure of a VNS heuristic. In that case the resulting VNS is called *two-level VNS*.

The variable neighborhood decomposition search (VNDS) [122] is based on decomposition of the problem. Namely, unlike other VNS variants VNDS does not launch an improvement phase in the whole solution space of a considered problem, but within a reduced solution space that corresponds to a reduced problem derived from the original one. The steps of VNDS are given in Algorithm 33. In this algorithm, the solution y corresponds to the solution obtained by decomposing the problem. Usually, it is generated so that it has k different attributes than the current incumbent solution x. On such generated, solution, an improvement phase is applied in the reduced solution space. Then, the solution returned by the improvement procedure is used to create the solution of the original problem. Namely, the solution returned by the improvement procedure together with the partial solution $x' \setminus y$ constitutes a solution of the original problem. As an improvement procedure within VNDS may be used some local search procedure, VND variant or some VNS variant.

Algorithm 33: Variable Neighborhood Decomposition Search

Function VNDS(x, k_{max}, $\mathcal{N}$)
repeat
  $k \leftarrow 1$;
  repeat
    $x' \leftarrow$ `Shake`$(x, k, \mathcal{N})$;
    $y \leftarrow x' \setminus x$;
    $y' \leftarrow$ `Improvement_procedure`(y);
    $x'' = (x' \setminus y) \cup y'$;
    `Neighborhood_change_sequential`(x, x'', k);
  until $k = k_{max}$;
until *stopping condition is fulfilled;*
return x;

The primal-dual VNS [125] is a variant of VNS that provides the estimation of quality of the obtained solution unlike the other VNS variants. Namely, the primal-dual VNS in the first phase uses a VNS heuristic to obtain a primal feasible solution. After that this solution is used to deduce a dual (infeasible)

solution. On a such obtained dual solution, a VNS heuristic is applied in order to reduce dual infeasibility followed by an exact method to solve relaxed dual problem and generate a lower bound. Finally, a standard branch-and-bound algorithm is launched to find an optimal solution of the original problem using tight upper and lower bounds, obtained from the heuristic primal solution and the exact dual solution, respectively.

It is well known that many optimization problems have more than one formulation. In the case that a problem has several formulations very often it appears that local optima with respect to one formulation is not necessarily a local optima with respect to another formulation. This fact has been exploited by *Variable neighborhood formulation space search* [168]. Its main idea is to use several formulations of a considered problem, and to switch from one to another formulation, depending on the current state of the solution process. When the change from one formulation to another occurs, the solution resulting from the former formulation is used as an initial solution for the latter formulation. In algorithm 34, one formulation change procedure is given. In this procedure, notation $f(\phi, x)$ corresponds to the objective value of the solution x in the formulation ϕ, while assignment $\phi \leftarrow \phi'$ represents the change of a formulation.

Algorithm 34: Formulation change procedure

Procedure `Formulation_change`(x, x', ϕ, ϕ',k)
if $f(\phi', x') < f(\phi, x)$ **then**
  $x \leftarrow x'$; $\phi \leftarrow \phi'$; $k \leftarrow 1$;
else
  $k \leftarrow k + 1$;
end

2.4 Variable neighborhood search applications

2.4.1 Traveling salesman problem with time windows

The Traveling Salesman Problem with Time Windows (TSPTW) is stated in the following way. Suppose that is given a depot and a set of customers

(each customer has its service time (i.e., the time that must be spent at the customer) and a time window defining its ready time and due date). The TSPTW problem consists in finding a minimum cost tour starting and ending on a given depot, visiting each customer exactly once before its due date. The traveling salesman is allowed to arrive at some customer before its ready time, but in that case the traveling salesman has to wait.

Since TSPTW is NP-hard problem, there is a need for a heuristic which will be able to solve efficiently realistic instances in the reasonable amount of time. In that direction, some steps have been already made. For example, Carlton and Barnes [33] had developed a tabu search heuristic with a static penalty function, that considers infeasible solutions. Gendreau et al. [82] had proposed a insertion heuristic based on GENIUS [82] that gradually builds the route and improves it in a post-optimization phase based on successive removal and reinsertion of nodes. Calvo [29] had suggested a heuristic which firstly solves an assignment problem with an ad hoc objective function and buildes a feasible tour merging all found sub-tours into a main tour, followed by a 3-opt local search procedure [138] to improve the initial feasible solution. Ohlmann and Thomas [173] developed a variant of simulated annealing, called compressed annealing, that relaxes the time windows constraints by integrating a variable penalty method within a stochastic search procedure. Two new heuristics were proposed in 2010. One heuristic was proposed by Blum et al. [155], while the other was proposed by Da Silva et al. [54]. These two heuristics are, currently, the state-of-the-art heuristics.

We propose new two stage VNS based heuristic for solving the TSPTW problem. At the first stage we use VNS to obtain a feasible initial solution, while at the second stage, we use GVNS to improve the initial solution obtained at the previous stage. Building an initial feasible solution is also a NP-hard problem. So, we decide to start with the solution obtained as in the procedure proposed in [54]. It is a VNS based procedure that relocates customers of a random solution (minimizing its infeasibility) until a feasible solution is obtained. We also tried out different usual initialization strategies, but they did not show better performances than one from [54].

Most common moves performed on a TSP solution are 2-opt moves and OR-opt moves [138]. The 2-opt move breaks down two edges of a current

solution, and inserts two new edges by inverting the part of a solution in such a way that the resulting solution is still a tour. One variant of the 2-opt move is the so-called 1-opt move which is applicable on four consecutive nodes. On the other hand, an OR-opt move relocates a chain of consecutive customers without inverting any part of a solution. If a chain contains k nodes, we call such move OR-opt-k move. If a chain of k consecutive nodes is moved backward, that move will be called backward OR-opt-k. Similarly, if a chain is moved forward, the move will be called forward OR-opt-k. Each of these moves defines one neighborhood structure of the given TSP solution as a set of all solutions obtained by performing that move on the given TSP solution.

The GVNS used in the second stage applies the cyclic VND to improve the initial solution. The neighborhood structures used within the cyclic VND are based on TSP moves such as: 1-opt, backward OR-opt-2, forward OR-opt-2, backward OR-opt-1, forward OR-opt-1, 2-opt (explored in the given order). Each of these neighborhood structures is explored by using the best improvement search strategy.

During a neighborhood exploration, it is important to check whether a move yields a feasible or an infeasible solution. For that purposes, we build an array g where g_i denotes maximal value for which arrival time at a node i, i.e., β_i, can be increased so that the feasibility on the final part of a tour, which starts at the node i, is kept. Elements of the array g are evaluated starting from the depot and moving backward through a tour. If we suppose that the node j precedes the node i, then g_j is calculated in the following way:

$$g_j = \min\{g_i + \max\{0, a_j - \beta_j\}, b_j - \beta_j\} \tag{2.2}$$

where $g_0 = b_0 - \beta_0$. Using the above array, if we want to check the feasibility of a move that involves all nodes between the node i and the node j (including them) from a tour, it suffices to calculate the new arrival time at each of those nodes as well as the new arrival time at the node k (the successor of the node j) that would occur if a move is executed. If all these new arrival times do not violate time window constraints and the arrival time at the node k, is increased for the value less or equal to the value of g_k, then a move is feasible otherwise it is infeasible.

The shaking procedure within GVNS performs k random feasible OR-opt-1 moves on a given solution.

Computational results

The proposed GVNS is run on a 2.53GHz processor (similar to one used in [54](2.4GHz processor)). The method is tested on the benchmark test instances from the literature. All test instances are grouped in test cases consisted of five test instances. The number of customers as well as the maximum range of time windows in each test instance can be deduced from the name of the test case to which that instance belongs. For example, each test instance in the test case n400w500 has 400 customers and maximum range of time window equal to 500. Since, the proposed method is stochastic one, we decide to run it 30 times on each test instance as Da Silva et al. [54] did. For each test case we calculate the average value of the objective function, the average time and the standard deviation σ. The obtained results are compared to those obtained GVNS proposed by Da Silva et al. [54] which is the state-of-the-art heuristic for considered instances.

Test instances proposed by Da Silva et al. [54]: The proposed GVNS with time limit set to 30 seconds has been tested on test cases proposed by Da Silva et al. According to obtained results (Table 2.1), the proposed GVNS offers 14 new best known solutions reducing the average computational time, in comparison to the GVNS proposed in [54], for about 50% in mean. Also, it should be noted that the proposed GVNS heuristic has not found the best known solution just for instance n300w200. However, the average value, 12142.71, obtained by the proposed GVNS on all test cases, is better than the average value, 12149.66, obtained by the GVNS proposed in [54].

Test instances proposed by Gendreau [82]: On all test instances proposed by Gendreau, we have run the GVNS with the time limit set to 10 seconds. Computational results obtained by the GVNS is presented in Table 2.2. According to those results, the proposed GVNS offers one new best known solution (test case n100w100), while on all other instances offers the same solution values as GVNS presented in [54]. Similarly as on

Test Case	GVNS			Time GVNS		GVNS [54]			Time GVNS [54]	
	min.value	av.value	σ	av.sec.	σ	min.value	av.value	σ	av.sec.	σ
n200w100	10019.60	10020.43	0.37	0.00	0.00	10019.6	10019.6	0.1	4.8	0.3
n200w200	9252.00	9254.23	4.99	0.13	0.06	9252.0	9254.1	7.2	5.8	0.2
n200w300	**8022.80**	8023.13	0.15	10.02	8.47	8026.4	8034.3	4.5	7.2	0.2
n200w400	**7062.40**	7072.36	9.21	11.84	8.50	7067.2	7079.3	4.4	8.7	0.4
n200w500	**6466.20**	6472.74	4.89	13.83	9.61	6466.4	6474.0	5.1	10.0	0.3
n250w100	12633.00	12633.00	0.00	0.01	0.01	12633.0	12633.0	0.0	9.9	0.2
n250w200	11310.40	11314.04	2.20	0.32	0.24	11310.4	11310.7	0.7	11.9	0.4
n250w300	10230.40	10231.06	1.49	3.70	4.33	10230.4	10235.1	2.8	14.9	0.6
n250w400	**8896.20**	8897.94	2.28	37.71	17.40	8899.2	8908.5	4.1	18.9	0.7
n250w500	**8069.80**	8083.47	5.33	42.23	17.65	8082.4	8082.4(!)	6.7	20.7	0.9
n300w100	15041.20	15041.20	0.00	0.01	0.01	15041.2	15041.2	0.0	21.2	0.7
n300w200	13851.40	13857.56	6.59	0.61	0.34	13846.8	13853.1	2.3	23.7	0.6
n300w300	**11477.20**	11478.84	1.17	10.97	6.92	11477.6	11488.5	5.2	37.0	3.8
n300w400	**10402.80**	10419.63	12.57	30.01	13.94	10413.0	10437.4	12.9	31.7	1.2
n300w500	**9842.20**	9849.23	3.54	49.48	15.72	9861.8	9876.7	8.9	35.4	1.1
n350w100	17494.00	17494.00	0.00	0.02	0.01	17494.0	17494.0	0.0	41.0	2.5
n350w200	15672.00	15672.00	0.00	1.66	2.23	15672.0	15672.2	0.6	47.3	2.1
n350w300	**13648.80**	13660.80	7.98	13.25	8.51	13650.2	13654.1	1.7	54.9	2.2
n350w400	**12083.20**	12090.56	4.60	46.83	13.54	12099.0	12119.6	8.9	60.2	2.8
n350w500	**11347.80**	11360.58	8.54	59.00	13.51	11365.8	11388.2	12.0	57.8	1.2
n400w100	19454.80	19454.80	0.00	0.01	0.00	19454.8	19454.8	0.0	57.1	0.6
n400w200	18439.80	18442.56	2.25	1.78	0.92	18439.8	18439.9	0.6	66.9	1.9
n400w300	**15871.80**	15875.83	3.95	28.78	11.20	15873.4	15879.1	3.0	93.6	7.9
n400w400	**14079.40**	14112.02	9.89	54.92	14.92	14115.4	14145.5	12.9	96.2	3.9
n400w500	**12716.60**	12755.80	14.25	77.46	16.40	12747.6	12766.2	9.7	109.3	4.4
Average	12135.43	12142.71	4.25	19.78	7.38	12141.58	12149.66	4.57	37.84	1.64

Table 2.1: Results on the test instances proposed by Da Silva et al.

previous test cases, the proposed GVNS is significantly faster than GVNS from [54] regarding mean time needed to solve one instance. According to

Test Case	GVNS			Time GVNS		GVNS [54]			Time GVNS [54]	
	min.value	av.value	σ	av.sec.	σ	min.value	av.value	σ	av.sec.	σ
n20w120	265.6	265.60	0.00	0.00	0.00	265.6	265.6	0.0	0.3	0.0
n20w140	232.8	232.80	0.00	0.01	0.00	232.8	232.8	0.0	0.3	0.0
n20w160	218.2	218.20	0.00	0.00	0.00	218.2	218.2	0.0	0.3	0.0
n20w180	236.6	236.60	0.00	0.00	0.00	236.6	236.6	0.0	0.4	0.0
n20w200	241.0	241.00	0.00	0.01	0.00	241.0	241.0	0.0	0.4	0.0
n40w120	377.8	377.80	0.00	0.02	0.01	377.8	377.8	0.0	0.8	0.0
n40w140	364.4	364.40	0.00	0.02	0.01	364.4	364.4	0.0	0.8	0.0
n40w160	326.8	326.80	0.00	0.02	0.01	326.8	326.8	0.0	0.9	0.0
n40w180	330.4	330.51	0.40	2.22	2.55	330.4	331.3	0.8	1.0	0.0
n40w200	313.8	313.83	0.14	3.60	2.70	313.8	314.3	0.4	1.0	0.1
n60w120	451.0	451.00	0.00	0.28	0.36	451.0	451.0	0.1	1.5	0.1
n60w140	452.0	452.00	0.00	0.09	0.09	452.0	452.1	0.2	1.7	0.1
n60w160	464.0	464.58	0.11	0.04	0.02	464.0	464.5	0.2	1.7	0.0
n60w180	421.2	421.20	0.00	0.45	0.38	421.2	421.2	0.1	2.2	0.1
n60w200	427.4	427.40	0.00	0.32	0.30	427.4	427.4	0.0	2.4	0.1
n80w100	578.6	578.60	0.00	0.73	0.84	578.6	578.7	0.2	2.3	0.1
n80w120	541.4	541.41	0.04	1.30	1.95	541.4	541.4	0.0	2.7	0.1
n80w140	506.0	506.28	0.26	1.37	1.04	506.0	506.3	0.2	3.2	0.3
n80w160	504.8	505.07	0.54	1.46	2.37	504.8	505.5	0.7	3.3	0.1
n80w180	500.6	500.91	0.99	3.33	2.48	500.6	501.2	0.9	3.7	0.1
n80w200	481.8	481.80	0.00	0.40	0.41	481.4	481.8	0.1	4.2	0.2
n100w100	640.6	641.02	0.64	2.79	2.40	642.0	642.1	0.1	3.7	0.1
n100w120	597.2	597.47	0.23	5.65	3.33	597.2	597.5	0.3	4.1	0.2
n100w140	548.4	548.40	0.00	0.23	0.07	548.4	548.4	0.0	4.4	0.2
n100w160	555.0	555.00	0.00	1.11	0.62	555.0	555.0	0.1	5.1	0.2
n100w180	561.6	561.60	0.00	1.22	1.78	561.6	561.6	0.0	6.3	0.3
n100w200	550.2	550.57	1.74	3.97	2.29	550.2	551.0	1.2	6.8	0.3
n100w80	666.4	666.40	0.00	0.46	0.38	666.4	666.6	0.2	3.1	0.2
Average	441.27	441.37	0.18	1.11	0.94	441.31	441.50	0.2	2.45	0.10

Table 2.2: Results on the test instances proposed by Gendreau

the computational results, the proposed GVNS is more efficient than the

previously proposed GVNS [54] regarding the quality of obtained solutions and consumed computational time. The efficiency of our implementation, in comparison to previously proposed GVNS([54]), relies on the larger number of neighborhood structures which have been examined as well as on the new efficient feasibility checking procedure. The larger number of neighborhood structures allows us to explore bigger part of the set of all feasible solutions of TSPTW problem and therefore to find good quality solutions. Naturally, exploring larger number of neighborhood structures is time wasting. However, reducing the size of neighborhood structures and using new feasibility checking procedure, we have been able to finish exploration quickly. In other words, using larger number of neighborhood structures, reducing neighborhood structures and introducing new way to check feasibility of some move, we have made compromise between solution quality and computational time to obtain that solution.

2.4.2 Attractive traveling salesman problem

Attractive traveling salesman problem (AtTSP) [67] is a variant of TSP. It is defined on an undirected graph $G = (V \cup W, E)$, where $V \cup W$ is a vertex set and $E = \{(v_i, v_j) : v_i, v_j \in V \cup W, i < j\}$ is an edge set. Each edge (v_i, v_j) from the set E has the associated distance d_{ij} as well as the travel time t_{ij}. The set V is a set of facility vertices, while the set W is a set of customer vertices. A subset of compulsory vertices of V is denoted by T. The set T contains at least the depot vertex, denoted by v_0. Each visited facility vertex, apart from the depot, generates a profit derived from customer vertices, which is measured in the way described later. The goal of AtTSP is to design a tour of maximal profit that contains all compulsory vertices and some vertices from $V \setminus T$. However, the length of a tour is limited by predefined upper bound L. The length of a tour includes travel times as well as dwell times associated with visited facility vertices.

The profit of a tour x is a sum of all profits achieved at each facility vertex that belongs to the tour apart from the depot. If we suppose that the facility vertex v_i is visited, than the profit generated by the customer vertex w_k at

v_i, i.e., P_{ki} is calculated in the following way:

$$P_{ki} = p_k \frac{\frac{a_i}{d_{ki}^2}}{b_k + \sum_{v_j \in V \setminus \{v_0\}} \frac{a_j}{d_{kj}^2} y_j}, \ \forall w_k \in W, \forall v_i \in V \tag{2.3}$$

where a_i is attractiveness of facility vertex v_i, b_k is self-attractiveness of customer vertex w_k, p_k is profit associated with customer vertex w_k, and y_j is equal to 1 if facility vertex v_j is visited, otherwise y_j is equal to 0. Therefore, the total profit of the tour x is equal to:

$$P(x) = \sum_{w_k \in W} \sum_{v_i \in V \setminus \{v_0\}} P_{ki} y_i \tag{2.4}$$

Application of the AtTSP arises in planning the tour of a mobile entertainment facility, such as the circus or the theater company, in routing military reconnaissance vehicles, in designing a route for a mobile health care facility operating in an underdeveloped region (see [67, 129, 176]). Since the AtTSP is a NP-hard problem [67], there is a need for heuristics able to provide good quality solutions very quickly. Some steps in that direction have already been made. Erdogan et al. [67] propose a tabu search for AtTSP, and GENIUS [81] for checking feasibility by solving TSP. Also, they propose a Branch and Cut (B&C) based exact algorithm.

We propose 2-level GVNS (2-GVNS) heuristic for solving AtTSP. Our 2-GVNS is actually a GVNS that uses another GVNS (the second level GVNS) to recognize feasible solutions.

Since the value of the objective function (2.4) does not depend on order of visited facility vertices the solution space examined within the first level GVNS is defined as $S = \{x | x = T \cup U, \forall U \subset V \setminus T$, such that vertices from x can constitute a tour whose length is less than or equal to $L\}$.

Since each feasible solution of AtTSP has to contain compulsory vertices, the initial solution for the first level is chosen as a set T, i.e., the set of compulsory vertices.

In the first level, the following neighborhoods of a given feasible solution x are explored:

- $Add(x) = \{x' | x' \text{ is feasible}, x \subset x', |x'| = |x| + 1\}$;
- $Drop(x) = \{x' | x' \text{ is feasible}, x' \subset x, |x'| = |x| - 1\}$;
- $Interchange(x) = \{x' | x' \text{ is feasible}, |x \triangle x'| = 2, |x'| = |x|\}$.

It is interesting to note that the above *Drop* neighborhood structure cannot improve the current objective function value. This result is given in the next proposition.

Proposition 2.4.1 *The value of the objective function (2.4) at each solution x' from neighborhood $Drop(x)$ is not greater than the value of the objective function at solution x.*

SeqVND in the first level systematically explores neighborhood structures *Add* and *Interchange* in that order. Due to Proposition 2.4.1, we do not perform *Drop* local search, as done in [67]. We implement the first improvement strategy in both neighborhood structures.

To diversify the search, we use unusual 'double' Shaking procedure. It firstly interchanges k random pairs of facilities and then, removes k facilities from the current solution x, also at random.

Since the neighborhood structures are defined as the sets of feasible solutions, we need a tool to recognize feasible solutions very quickly. The only way to recognize vertices of a given solution as those that can form the tour, whose length does not exceed a predefined limit L, is to solve the associated TSP, which is NP-hard. For that purposes, we develop GVNS heuristic in the second level. Neighborhood structures examined within SeqVND of the second level GVNS are 2.5_opt [138] and so-called 1_opt neighborhood structures. The shaking function in the second level preforms k random 2.5_opt moves on a given tour x.

Computational results

Our 2-GVNS is run on a computer with 2.53 GHz CPU and 3GB of RAM, and has been tested on random test instances and extended TSP-Lib instances from [67]. All test instances from [67] are available at `http://neumann.hec.ca/chairelogistique/data/AtTSP/`.

TSP-Lib Instances [67]. First, we compare 2-GVNS with B&C algorithm [67], implemented by using CPLEX 10.0.1, to check the precision of our heuristic. Maximum time allowed for B&C algorithm was set to 1 hour (3600 seconds), while for 2-GVNS was set to 300 seconds.

In Table 2.3, results on the same instances as in [67] are presented. They contain 6 different files from TSP-Lib: *kroA, kroB, kroC, kroD, kroE* and *ali535*. These 6 files are used to create 48 test instances choosing different values for $|V|, |W|$ and $|T|$. The column headings are defined as follows:

- $|V|, |T|, |W|$ - cardinalities of sets V, T and W, respectively;
- *B&C*- the value obtained by B&C algorithm;
- *GVNS* - the value of the solution found by 2-GVNS;
- *% dev*- percentage deviation of 2-GVNS from the corresponding B&C value calculated as

$$\% \; dev = \frac{f_{\texttt{B\&C}} - f_{\texttt{GVNS}}}{f_{\texttt{B\&C}}} \times 100.$$

- *time*- time when the 2-GVNS (B&C) solution is reached.
- *size*- the number of nodes in the 2-GVNS solution.

Note that on some instances, objective function values reported by B&C and 2-GVNS are slightly different although the solutions are the same. This phenomenon can be explained by the fact that all variables in the objective function have type double.

It appears:

- Our 2-GVNS reported the same objective function values as B&Con all 48 instances, but in much less computing time. This means that the exact solutions of all 47 instances were reported (note that B&C algorithm was able to find optimal solutions for all TSP-Lib instances, except for the instance ali535, with $|V| = 50, |T| = 1, |W| = 150$);

Test Instance	$\|V\|$	$\|T\|$	$\|W\|$	Objective func. B&C	Objective func. GVNS	(%) dev	Time (sec) B&C	Time (sec) GVNS	Size
kroA100	25	1	75	3208.87	3208.82	0.00	34.06	10.56	11
kroA100	25	6	75	3415.33	3415.31	0.00	8.16	0.72	15
kroA100	25	12	75	3491.10	3491.08	0.00	1.72	0.11	19
kroA100	25	18	75	3562.48	3562.47	0.00	0.06	0.01	24
kroA150	37	1	113	4781.18	4781.12	0.00	381.00	2.62	12
kroA150	37	9	113	5083.13	5083.07	0.00	106.79	1.22	17
kroA150	37	18	113	5319.77	5319.73	0.00	25.27	1.52	24
kroA150	37	27	113	5417.65	5417.59	0.00	1.42	0.02	31
kroA200	50	1	150	5571.03	5570.95	0.00	2047.76	5.51	13
kroA200	50	12	150	5777.59	5777.52	0.00	83.27	1.05	18
kroA200	50	25	150	6334.22	6334.16	0.00	11.13	0.74	28
kroA200	50	37	150	6584.07	6583.97	0.00	3.75	0.03	38
kroB100	25	1	75	3074.84	3074.80	0.00	1.35	2.03	12
kroB100	25	6	75	3046.43	3046.40	0.00	5.08	2.17	14
kroB100	25	12	75	3149.02	3148.97	0.00	0.32	0.18	19
kroB100	25	18	75	3194.25	3194.22	0.00	1.15	0.20	23
kroB150	37	1	113	4278.50	4278.45	0.00	513.81	4.56	12
kroB150	37	9	113	4553.95	4553.90	0.00	42.83	0.81	17
kroB150	37	18	113	4775.70	4775.66	0.00	2.12	0.46	23
kroB150	37	27	113	4907.78	4907.72	0.00	0.99	0.10	30
kroB200	50	1	150	5158.18	5158.12	0.00	147.86	8.03	13
kroB200	50	12	150	5122.27	5122.22	0.00	627.50	1.25	19
kroB200	50	25	150	5725.12	5725.04	0.00	10.31	1.03	30
kroB200	50	37	150	6001.81	6001.77	0.00	2.25	0.06	39
kroC100	25	1	75	2910.12	2910.08	0.00	10.48	0.54	11
kroC100	25	6	75	3055.04	3055.00	0.00	2.86	0.81	15
kroC100	25	12	75	3119.39	3119.35	0.00	2.84	0.24	17
kroC100	25	18	75	3237.36	3237.34	0.00	0.32	0.03	22
kroD100	25	1	75	3002.30	3002.27	0.00	8.15	0.58	13
kroD100	25	6	75	3094.04	3094.01	0.00	0.89	0.28	16
kroD100	25	12	75	3146.34	3146.32	0.00	0.14	0.09	19
kroD100	25	18	75	3170.25	3170.24	0.00	0.04	0.02	23
kroE100	25	1	75	3311.27	3311.23	0.00	3.77	2.78	12
kroE100	25	6	75	3313.15	3313.13	0.00	0.91	1.13	14
kroE100	25	12	75	3393.93	3393.90	0.00	2.18	0.14	19
kroE100	25	18	75	3474.43	3474.42	0.00	0.11	0.01	24
ali535	25	1	75	2697.01	2696.97	0.00	0.29	1.28	18
ali535	25	6	75	2721.78	2721.76	0.00	0.08	0.22	22
ali535	25	12	75	2725.05	2725.05	0.00	0.04	0.06	24
ali535	25	18	75	2729.37	2729.36	0.00	0.13	0.00	25
ali535	37	1	111	3170.43	3170.37	0.00	55.19	4.76	19
ali535	37	9	111	3370.52	3370.48	0.00	4.15	2.74	25
ali535	37	18	111	3427.33	3427.28	0.00	1.20	0.63	29
ali535	37	27	111	3515.70	3515.63	0.00	0.49	0.09	34
ali535	50	1	150	3660.21	3660.15	0.00	3600.30	12.88	19
ali535	50	12	150	3682.20	3682.13	0.00	39.29	4.08	24
ali535	50	25	150	4127.52	4127.46	0.00	38.72	1.49	35
ali535	50	37	150	4413.44	4413.36	0.00	1.95	0.16	43
Average				3958.38	3958.34	0.00	163.22	1.67	21.31

Table 2.3: Computational results on TSPLIB instances

- On the other hand, as reported in [67], Tabu Search algorithm failed to find optimal solutions for the following instances: kroA100 $|V| = 25, |T| = 1, |W| = 75$, kroA150 $|V| = 37, |T| = 9, |W| = 113$, kroB150 $|V| = 37, |W| = 113, |T| = 1$ and 9, kroB200 $|V| = 50, |W| = 150, |T| = 12$ and 25, ali535 $|V| = 37, |W| = 111, |T| = 1$, 9, and 18, $|V| = 50, |T| = 1, |W| = 150$. Thus, we may conclude that 2-GVNS is the best method for solving this type of instances.

Test instances for extended AtTSP [67]. Extension of AtTSP considers more than one service option for visiting a facility vertex. These options may differ in the service time length and the attractiveness values. The instances of extended AtTSP are transformed to AtTSP instances, in the way described in [67]. For this purpose, the kroA, kroB, kroC, kroD and kroE instances were used, producing 36 instances in total. The comparison of 2-GVNS with `B&C` and Tabu Search (TS) methods is given in Table 2.4. The new column headings are defined as follows:

- *TS*- objective value of the solution found by TS,
- % dev *TS*- deviation of `GVNS` from the corresponding TS value calculated as $\frac{f_{TS}-f_{GVNS}}{f_{TS}} \times 100$.

In our early experiments, we were not able to reach results reported in [67] for some instances. We asked authors to provide us with optimal solutions of those instances. They have kindly sent us new results, which are reported in column 5 of Table 2.4. This time, their maximum time spent in solving one instance has increased to 2 hours (see column 12 of Table 2.4). Note that, in the original paper, maximum time allowed was 1 hour. Note also that the new solutions, as well as our solutions, can be found at `http://www.mi.sanu.ac.rs/~nenad/attsp/`.

The presented results show that the 2-GVNS improves the best known solutions for 2 instances (out of 36): kroB150 $|V| = 37, |T| = 1, |W| = 113$ and kroA200 $|V| = 50, |T| = 1, |W| = 150$, while on all other instances 2-GVNS is able to find same solutions as `B&C` algorithm. However, 2-GVNS needs significantly less computational time. When compared with TS algorithm, the proposed 2-GVNS provides significantly better solutions on almost all test instances. Moreover, there is no test instance on which TS performs better than 2-GVNS. This can be explained by the fact that TS and 2-GVNS examine different neighborhood structures. The TS heuristic explores *Drop* and *Add*, while 2-GVNS examines *Add* and *Interchange* neighborhoods. More precisely, since *Drop* neighborhood does not contain improvement of a solution, 2-GVNS explores larger part of the solution space and therefore has more chances to find a better solution than TS. Consequently, 2-GVNS

Test				Objective function value			(%) dev GVNS vs.		Size		Time (sec)	
Instance	$\|V\|$	$\|T\|$	$\|W\|$	B&C	TS	GVNS	TS	B&C	B&C	GVNS	B&C	GVNS
kroA100	25	1	75	2356.96	2321.32	2356.92	-1.53	0.00	9	9	1622.75	83.09
kroA100	25	6	75	2588.61	2576.96	2588.57	-0.45	0.00	10	10	137.82	82.80
kroA100	25	12	75	2725.51	2723.86	2725.48	-0.06	0.00	16	16	42.80	8.72
kroA100	25	18	75	2879.15	2877.97	2879.10	-0.04	0.00	20	20	135.96	1.73
kroA150	37	1	113	3516.82	3490.45	3516.94	-0.76	0.00	8	8	3867.79	241.33
kroA150	37	9	113	3882.45	3853.74	3882.40	-0.74	0.00	14	14	75.73	34.36
kroA150	37	18	113	4166.33	4166.33	4166.28	0.00	0.00	20	20	8.05	6.60
kroA150	37	27	113	4268.36	4268.37	4268.31	0.00	0.00	30	30	16.58	3.67
kroA200	50	1	150	3775.93	3636.83	**3781.07**	-3.97	-0.14	9	10	7200.73	68.17
kroA200	50	12	150	3938.36	3910.39	3938.29	-0.71	0.00	17	17	868.07	20.44
kroA200	50	25	150	4545.32	4545.33	4545.28	0.00	0.00	28	28	321.16	5.71
kroA200	50	37	150	4914.69	4849.82	4914.63	-1.34	0.00	38	38	1.67	0.90
kroB100	25	1	75	2444.09	2442.18	2444.05	-0.08	0.00	8	8	48.29	46.97
kroB100	25	6	75	2392.91	2392.91	2392.88	0.00	0.00	11	11	8.70	42.56
kroB100	25	12	75	2507.47	2507.46	2507.43	0.00	0.00	15	15	2.32	37.22
kroB100	25	18	75	2599.72	2599.71	2599.68	0.00	0.00	20	20	18.85	34.42
kroB150	37	1	113	3005.74	2948.05	**3013.83**	-2.23	-0.27	8	9	7235.95	25.87
kroB150	37	9	113	3282.67	3272.50	3282.61	-0.31	0.00	15	15	41.91	19.46
kroB150	37	18	113	3525.58	3525.57	3525.53	0.00	0.00	22	22	18.24	10.97
kroB150	37	27	113	3700.56	3700.55	3700.50	0.00	0.00	29	29	3.45	2.60
kroB200	50	1	150	3561.77	3515.09	3561.76	-1.33	0.00	10	10	3877.27	40.86
kroB200	50	12	150	3537.64	3528.10	3537.58	-0.27	0.00	20	20	252.72	68.24
kroB200	50	25	150	4209.81	4209.40	4209.76	-0.01	0.00	30	30	163.37	17.72
kroB200	50	37	150	4597.73	4597.72	4597.69	0.00	0.00	38	38	10.33	3.40
kroC100	25	1	75	2114.74	2039.45	2114.70	-3.69	0.00	8	8	166.30	45.98
kroC100	25	6	75	2287.62	2234.55	2287.60	-2.37	0.00	10	10	53.78	25.73
kroC100	25	12	75	2303.61	2303.60	2303.57	0.00	0.00	15	15	7.81	6.78
kroC100	25	18	75	2524.54	2491.42	2524.49	-1.33	0.00	18	18	0.26	2.60
kroD100	25	1	75	2335.90	2306.69	2335.86	-1.26	0.00	8	8	158.82	76.47
kroD100	25	6	75	2435.53	2422.13	2435.51	-0.55	0.00	12	12	40.72	123.76
kroD100	25	12	75	2522.18	2494.43	2522.16	-1.11	0.00	15	15	0.47	8.88
kroD100	25	18	75	2642.83	2642.82	2642.78	0.00	0.00	19	19	1.13	3.86
kroE100	25	1	75	2618.70	2526.25	2618.66	-3.66	0.00	7	7	10.73	26.55
kroE100	25	6	75	2561.25	2492.11	2561.22	-2.77	0.00	10	10	1.74	11.49
kroE100	25	12	75	2659.51	2651.28	2659.50	-0.31	0.00	16	16	1.08	9.84
kroE100	25	18	75	2766.33	2766.34	2766.30	0.00	0.00	22	22	10.00	10.22
Average				3130.47	3106.44	3130.80	-0.86	-0.01	16.81	16.86	734.26	35.00

Table 2.4: Computational results on extended AtTSP instances

succeeded to cope with test instances that were very difficult for TS. In addition, the feasibility checking phase of GVNS, applied at the second level, appears to be very efficient.

2.4.3 Traveling salesman problem with draft limits

Recently, new variant of Traveling salesman Problem (TSP) in the context of maritime transportation, called Traveling Salesman problem with Draft Limits (TSPDL) has been proposed [186]. TSPDL consists of visiting and delivering goods for a set of ports using a ship located initially at a depot. Since each port has a delivery demand known in advance the ship starts the

tour with a load equal to the total demand, visits each port exactly once and comes back to the depot performing the lowest cost tour. However, to each port it is assigned a draft limit, which represents the maximal allowed load on the ship upon entering some port.

Formally, the problem may be stated as follows [186]. Given an undirected graph $G = (V, E)$ where $V = \{0, 1, ..., n\}$ represents the set of ports including the starting port, i.e., the depot denoted by 0, while $E = \{(i, j)|i, j \in V, i \neq j\}$ represents the edge set where each edge (i, j) from the set E has the associated cost c_{ij}. For each port i, apart from the depot, a draft limit, l_i, and a demand, d_i, are given. Therefore the goal of TSPDL is to design minimal cost Hamiltonian tour, visiting each port exactly once and respecting draft limit constraints of all ports. Additionally, let us denote by L_i the load on the ship upon entering the port i, calculated relatively to a given tour T. The tour T will be called a feasible tour if $L_i \leq l_i$ for all $i \in V$, otherwise it will be called an infeasible tour.

TSPDL is a NP-hard problem. Rakke et al. [186] proposed two formulations of TSPDL and a Branch and cut algorithm for solving both of them. Additionally, they introduced some valid inequalities and strengthened bounds that significantly reduce both the number of branch-and-bound nodes and the solution time. However, with these approaches they did not succeed to solve all instances to optimality. Recently, Battarra et al. [14] proposed three new formulations of TSPDL which enabled them to solve instances to optimality using exact algorithms. Up to now, just exact methods have been considered for solving this problem. However, it turns out that these methods consume a lot of CPU time to solve benchmark instances either to optimality or near optimality. Therefore, there is a need for metaheuristics or heuristic able to find near-optimal solutions quickly and also able to solve instances of larger size. We propose two heuristics based on GVNS for the TSPDL, which differ in a strategy of choosing the next neighborhood, from a given list, in the deterministic part of GVNS. One of them, named GVNS-1, uses sequential VND in the improvement phase, while another, named GVNS-2, uses cyclic VND.

It is obvious that not every solution of TSP, i.e., permutation of ports, is feasible for TSPDL and therefore some permutations might be infeasible.

However, the following proposition, proved in [186], provides a necessary and sufficient condition for feasibility.

Proposition 2.4.2 *Let suppose, without loss of generality, that all ports are sorted in descending order regarding draft limits such that* $l_i \leq l_{i+1}$, $\forall i \in \{1, \dots n-1\}$. *Then a TSPDL instance is feasible if and only if* $L_i \leq l_i$ *for all* $i \in V$. *If the solution* $(0, \dots, n)$ *is infeasible than no other solution is feasible.*

This proposition provides a way how to generate a first feasible solution if any exists. More precisely, to do this we have to sort the ports in descending order relatively to draft limits, and take this permutation as an initial solution if it is feasible. Any neighborhood structure applicable for TSP can be also adapted for TSPDL problem taking into account the draft limit constraints. This adaption may be performed in a simple way, by excluding all permutations which are infeasible for TSPDL. In order to quickly recognize infeasible solutions for each used neighborhood structures, we implement the feasibility checking function. In what follows, we describe the used neighborhood structures and the feasibility checking procedures used within them.

In order to recognize whether some move is feasible or infeasible, without performing it, we just have to keep array L updated and check whether the feasibility will be kept on some part of a tour if we execute that move.

Two variants of Variable Neighborhood Descent (VND), which systematically explore different neighborhood structures using the first improvement strategy, are proposed. The first VND is SeqVND which examines respectively 1-opt, 2-opt, backward OR-opt-3, forward OR-opt-3, backward OR-opt-2, forward OR-opt-2, backward OR-opt-1 and forward OR-opt-1 neighborhood structures. The second one is cyclic VND which examines respectively 1-opt, backward OR-opt-2, forward OR-opt-2, backward OR-opt-1, forward OR-opt-1 and 2-opt neighborhood structures in that order. Apart of using different neighborhood structures, these two variants of VND use different rules for choosing how to continue search after finishing search in some neighborhood structure. While SeqVND continues search in the first neighborhood structure or in the next neighborhood structure depending on whether some

improvement has been found or not, cyclic VND always continues search in the next neighborhood structure. Therefore these two variants have different stopping conditions. SeqVND stops when there is no improvement in the last neighborhood structure, i.e., forward OR-opt-1 neighborhood structure, while cyclic VND stops when there is no improvement between two consecutive iterations.

Since we have developed two VND procedures, we implemented two GVNS methods as well. The only difference between the GVNS methods is a different VND used as a local search. Depending on which VND is employed as a local search we denote these two GVNS methods as GVNS-1 (local search - SeqVND) and GVNS-2 (local search - cyclic VND). In order to escape from the current local optima both GVNS methods use the same shaking function which returns a solution obtained by preforming k times a random OR-opt-1 move on a given tour. More precisely the shaking function at each iteration chooses at random one port from the tour and moves it, either forward or backward, after another port, also chosen at random.

Computational results

For testing the proposed two approaches, we used the set of benchmark instances proposed in [186]. It consists of 240 instances derived from 8 classical TSP instances [189]. Each instances is characterized by a matrix of edge costs, a number of ports, demands and draft limits. They can be found at : http://jgr.no/tspdl. Furthermore, we generate 60 new large-size data sets for the TSPDL. The new ones are generated from three existing TSP instances [189] with $n \in \{100, 200, 442\}$. The new instances are available at : http://turing.mi.sanu.ac.rs/~nenad/TSPDL/ The time limit for both GVNS heuristics has been set to 100 seconds.

In Table 2.5, a summary results that compare our heuristics with the exact one based on *branch and cut and price* (B&C&P) provided in [186] are presented. Note that the B&C&P algorithm succeeded to solve all 240 instances to optimality within a time limit set to 7200 seconds. Table 2.5 reports in the first column the instance name while in the second column it reports the average of 10 optimal solutions found by the B&C&P algorithm.

The average times for all three methods are also given in Table 2.5. It appears: *i)* GVNS-2 heuristic reaches all 240 optimal solutions with 1.06 seconds on average. Note that the B&C&P used 4112.04 seconds. *ii)* GVNS-1, is slightly faster than GVNS-2 (it spends 0.82 seconds on average), but it did not get optimal solution in only 2 (out of 240) instances.

Inst	B&C&P		GVNS-1		GVNS-2	
	Optimal	Time	Value	Time	Value	Time
Burma_14_10	3386.70	0.52	3386.70	0.00	3386.70	0.00
Burma_14_25	3596.80	0.37	3596.80	0.00	3596.80	0.00
Burma_14_50	3862.30	0.35	3862.30	0.00	3862.30	0.00
Ulysse_16_10	6868.20	50.34	6868.20	0.00	6868.20	0.00
Ulysse_16_25	7165.40	18.58	7165.40	0.00	7165.40	0.00
Ulysse_16_50	7590.30	4.60	7590.30	0.00	7590.30	0.00
Ulysse_22_10	7087.60	32.02	7087.60	0.04	7087.60	0.00
Ulysse_22_25	7508.70	24.61	7508.70	0.01	7508.70	0.00
Ulysse_22_50	8425.60	26.55	8425.60	0.04	8425.60	0.03
fri26_25	963.80	12.37	963.80	0.02	963.80	0.00
fri26_50	1104.70	16.43	1104.70	0.06	1104.70	0.05
fri26_10	1178.70	16.26	1178.70	0.01	1178.70	0.01
bayg29_10	1713.60	11.99	1713.60	0.00	1713.60	0.02
bayg29_25	1792.60	11.05	1792.60	0.14	1792.60	0.02
bayg29_50	2091.00	13.38	2091.00	0.01	2091.00	0.05
gr17_10	2150.30	25.12	2150.30	0.00	2150.30	0.00
gr17_25	2237.70	11.11	2237.70	0.00	2237.70	0.00
gr17_50	2710.30	5.54	2710.30	0.00	2710.30	0.00
gr21_10	2833.60	11.63	2833.60	0.00	2833.60	0.01
gr21_25	2962.60	11.44	2962.60	0.00	2962.60	0.00
gr21_50	3738.10	10.32	3738.10	0.01	3738.10	0.00
gr48_10	5284.40	1352.72	5284.40	0.20	5284.40	3.39
gr48_25	5800.30	861.47	5802.60	13.25	5800.30	13.65
gr48_10	6635.70	211.35	6635.70	5.97	6635.70	8.31
Average	4112.04	114.17	4112.14	0.82	4112.04	1.06

Table 2.5: Summary of the Comparison between the existing method and the proposed two GVNS approaches on small instances

In Table 2.6, a comparison between the two proposed GVNS methods on new instances is presented. For each method, we calculate percentage deviation between average value of its solutions and the Lower Bound (LB), i.e., the optimal solution for TSP. Thus, we calculate the percentage deviation Δ as follows : $\Delta = ((f - LB)/LB) \times 100$, where f is the value of the objective function.

Inst	LB	GVNS-1		GVNS-2		Δ_{GVNS-1}	Δ_{GVNS-2}
		Value	Time	Value	Time		
KroA100_50	21282	21325.60	16.92	21294.00	27.86	0.20	0.06
KroA100_75	21282	21359.40	29.41	21303.40	32.67	0.36	0.10
KroA200_50	29368	30869.30	66.00	30665.20	77.51	5.11	4.42
KroA200_75	29368	32041.50	79.37	30896.10	77.05	9.10	5.20
pcb442_50	50778	61170.30	77.51	59858.30	72.74	20.47	17.88
pcb442_75	50778	63889.70	72.32	61010.10	83.36	25.82	20.15
Average	33809.33	38442.63	56.92	37504.52	61.87	10.18	7.97

Table 2.6: Comparison between the proposed two GVNS approaches on new large instances

From Table 2.6, we conclude that GVNS-2 outperforms GVNS-1 regarding average values of provided solution, although it consumes slightly greater amount of CPU-time. Note that the average percentage deviation increases as number of vertices increases. It should be emphasized that on some instances with 100 vertices GVNS variants succeeded to reach solutions equals to lower bound value. In other words, some instances with 100 vertices have been solved to optimality.

2.4.4 Swap-body vehicle routing problem

Swap-Body Vehicle Routing Problem (SBVRP) [218] is stated as follows. Given a homogeneous unlimited fleet located at the depot. The fleet contains trucks, trailers and swap-bodies. In the fleet, only swap bodies may be loaded but their maximum capacity Q must not be exceeded. From the available fleet, two vehicle combinations are allowed to be made: truck carrying one swap-body (SB) or truck and semi-trailer (called *train*) carrying two swap-bodies. Using available vehicle combinations one has to service a given set of customers visiting each of them exactly once while performing route that starts and ends at the depot. Each customer i has a demand q_i that must be delivered, servicing time s_i needed to serve it as well as the accessibility constraint. Namely, some customers (called *truck customers*) can be visited only by a truck while the others (called *train customers*) may be visited either by a truck or by a train. However, even if the truck departs from the depot as a train carrying on two swap-bodies it can visit truck customers

after detaching the semi-trailer at specified locations called *swap-locations*. If the semi-trailer is detached at some swap-location from the truck, it must be reattached by the same truck before returning to the depot. Note that no exchange of swap-bodies between trucks is allowed on route, i.e., a truck/train has to return to the depot with exactly the same swap-bodies it departed with. At the swap locations, it is allowed to perform the following actions:

- **Park action** - a truck parks the semi-trailer and continues the route with just one swap-body.
- **Pick Up action** - the truck picks up the semi-trailer with the swap-body it has parked there at an earlier stage of the route.
- **Swap action** - a truck parks the swap-body it is currently carrying and picks up the other swap-body from the semi-trailer it has parked there at an earlier stage of the route.
- **Exchange action** - a truck parks the semi-trailer, exchanges the swap-bodies between truck and semi-trailer and continues the route.

Note that it is not allowed to exchange load between swap-bodies neither at swap locations nor at the customers sites even if they belong to the same train. However, if a customer is visited by a train, its demand may be loaded on two swap-bodies.
Each performed route r has three attributes: the total distance, D_r^T, traveled by truck in kilometers (km), the total distance D_r^S traveled by semi-trailer in km, and total duration, τ_r, in hours. The route duration includes not only time needed to travel from one location to another in the route, but also the time spent serving customers on the route as well as time spent performing actions (if any) at swap locations. However, the duration of each route must not exceed the predefined time limit τ_{max}. Three aforementioned attributes together with the fixed cost for using a truck, C_F^T, and the fixed cost for using a semitrailer, C_F^S, induces the total cost of a route. More precisely, let C_D^T and C_D^S be cost per km traversed by a truck and a semi-trailer, respectively and let C_T be cost for one hour spent on a route. Then, the cost of a route r

performed by train is given as:

$$C_r = C_D^T \times D_r^T + C_D^S \times D_r^S + C_T \times \tau_r + C_F^T + C_F^S \tag{2.5}$$

Similarly, the cost of a route r performed by truck is calculated as:

$$C_r = C_D^T \times D_r^T + C_T \times \tau_r + C_F^T \tag{2.6}$$

So, the objective of SBVRP is to find a set of routes of minimum total cost such that in each route customer requirements, the allocated vehicle constraints, and maximal route duration constraint are respected.

We represent a solution of SBVRP as a set of routes. Each route is composed of a main tour and a set of sub-tours (if any). A main tour starts and ends at the depot while a sub-tour starts and ends at some swap-location visited in a main tour (see Figure 2.1). If a set of sub-tours is non-empty, then each sub-tour is rooted at some swap-location. Further in that case, each sub-tour contains either truck or train customers, while main-tour contains only train customers (if any) and swap-locations. Existence of a sub-tour in a route means that the train visits a swap-location, performs some swap-action (park or exchange), serves a set of customers (truck or train customers), returns to the last visited swap-locations and again performs some swap-action (pick-up or swap). In the case, that pick-up action occurs, the train continues traversing the main-tour, while otherwise, the truck makes one more sub-tour rooted at the same swap-location. On the other hand, in the case that the set of sub-tours is empty, we distinguish two types of main tour: a train main tour and a truck main tour. A train main tour is achieved visiting train customers using a train vehicle. On the other hand, a truck main tour arises when a truck is used to serve only truck customers, or only train customers, or both truck and train customers.

In order to construct an initial solution we used a cluster-first, route-second approach. In the first step, truck customers are clustered with respect to the depot and the swap locations which are considered as centers of clusters. The proximity of each truck customer i to a center j, δ_{ji} is estimated as a cost that will occur if the truck customer i is serviced via the center j. In the case that, the depot is considered as a center, the cost is estimated as a cost of

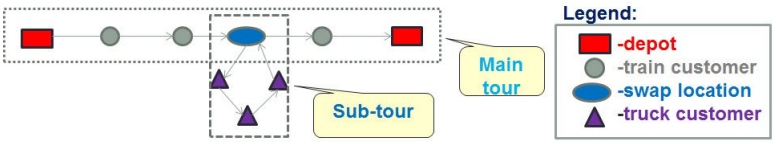

Figure 2.1: Solution representation

the tour performed by a truck going from the depot to a truck customer i and returning to the depot, i.e.,

$$\delta_{0i} = C_F^T + C_D^T \times (d_{0i} + d_{i0}) + C_T \times (t_{0i} + t_{i0} + s_i). \tag{2.7}$$

On the other hand, if a truck customer is served via a swap-location, the cost is calculated as the cost of the route traversed by a train vehicle going from the depot to a swap-location j (parking semi-trailer), visiting a truck customer i, coming back to the swap-location (picking up semi-trailer) and finally coming back to the depot, i.e.,

$$\delta_{ji} = C_F^T + C_F^S + C_D^T \times (d_{0j} + d_{ji} + d_{ij} + d_{j0}) + C_D^S \times (d_{0j} + d_{j0}) + C_T \times (t_{0j} + t_{ji} + t_{ij} + t_{j0} + s_i + \rho_P + \rho_U). \tag{2.8}$$

After clustering truck customers, the routing is performed in each cluster in order to create sub-tours as well as to create truck main tours. In order to accomplish that, in each obtained cluster we solve VRP using Clarke and Wright heuristic [42] assuming that we use only truck vehicles which capacity equal to Q, and setting the cluster center to serve as a depot in the considered VRP. Each route constructed in a cluster whose center is a swap-location represents a sub-tour (in our representation), while each route constructed in the cluster whose center is the depot represents a truck main tour.

In the last step, main tours are constructed. Main tours are obtained as solutions of VRP in which train customers and swap-locations are taken into account. To each swap-location is assigned demand equal to the total demand on the sub-tour (constructed in previous step) originating at it. Note that in the case that there is more than one sub-tour rooted at some swap-location, we create copies of the swap-location and set the demand of each copy to be equal to the total demand of one sub-tour rooted at the swap-location.

The resulting VRP is again solved by Clarke and Wright heuristic taking into account that we have the fleet of vehicles located at the depot which capacity is equal to $2Q$. In addition, we imposed the constraint that the total demand of swap-location nodes included in one route must not be greater than Q. Under such constraint the total demand of customers in the sub-tours will be loaded in a swap-body carried on a truck. The main purpose of this constraint is to ensure that the total demand of customers in the sub-tours will be satisfied.

For a solution of SBVRP, we define neighborhood structures adapting ones most commonly used in VRP related problems. The moves that are used to constitute these neighborhoods may be divided into two groups: Intra route moves (that involve one main tour or sub-tour) and inter-route moves (that involve two tours (either main tours or sub-tours)). The intra route moves that we used are actually standard traveling salesman problem (TSP) moves [138] and they are: 2-opt move, 1-opt move and OR-opt-1 move.

Inter - route neighborhood moves are classified into two groups according to whether they include the relocation of a part of one tour to another (Figure 2.2) or the exchange of two parts of two different tours (Figure 2.3). These moves are further classified with respect to the tours involved in a move. Therefore we distinguish the following eight moves: relocate a part from one main tour to another (`Relocate_main_to_main`); relocate a part from a main tour to a sub-tour (`Relocate_main_to_subtour`); relocate a part from a sub-tour to a main tour (`Relocate_subtour_to_main`); relocate a part from one sub-tour to another (`Relocate_subtour_to_subtour`); exchange parts of two main tour (`Exchange_main_main`); exchange parts of a main tour and a sub-tour (`Exchange_main_subtour`); exchange parts of two sub-tours (`Exchange_subtour_subtour`).

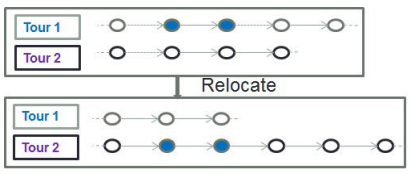

Figure 2.2: Relocate move

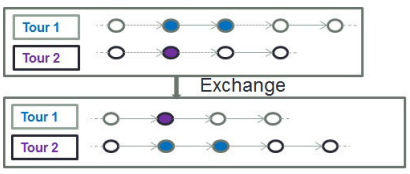

Figure 2.3: Exchange move

Note that in these moves some restrictions are imposed in order to preserve feasibility. Namely, we do not allow relocation of a part of a main tour which contains swap locations into a sub-tour. Similarly, a part of sub-tour that contains truck customers can not be relocated in a main tour if the resulting route would be served by a train. The similar restrictions are included in moves that include exchange of tour parts as well. It should be emphasized that relocate moves (`Relocate_subtour_to_main` and `Relocate_subtour_to_subtour`) also allow a movement of whole sub-tours and therefore sub-tour elimination. Similarly, `Exchange_subtour_subtour` move enable us to exchange two sub-tours within same route or between two different routes. In order to avoid additional computation needed for the feasibility checking, we treat as feasible moves only those which execution lead to routes such that the total demand on their sub-tours is not greater than Q. In other words, our solution space contains only solutions such that if a route in a solution contains a swap location, at this swap location only park and pick up actions are executed.

The previously described neighborhood structures have been exploited within VNS scheme. More precisely, the neighborhoods based on the similar move structures are embedded within the same VND scheme. In that way we create three seqVNDs:

- VND_TSP - explores neighborhoods of a given solution S that are based on standard TSP moves.
- VND_relocate - uses the neighborhoods of a given solution S that involves relocation of a part of one tour to another.
- VND_exchange - explores neighborhoods of a given solution S obtained exchanging the parts of tours.

Each of these VNDs uses the first improvement search strategy, i.e., as soon as a VND detects solution better than the current one, it resumes search starting from that solution. The presented VNDs are used as ingredients of a local search procedure used within the proposed GVNS. The proposed local search applies VND_TSP, VND_exchange and VND_relocate in that order one after another. The used shaking procedure has two parameters: solution S and the number of iterations, k, to be performed within it. In each iteration

the procedure firstly choose a tour (main or sub-tour) at random and then choose its part of random length such that a swap-locations is not included in the chosen part. After that each customer from the selected part is moved in another tour, keeping the feasibility and deteriorating the current objective value as least as possible. The such obtained solution is set to be the new incumbent solution S and whole process is repeated until reaching maximum number of iterations allowed (i.e., k).

Besides, the sequential GVNS we implemented a parallel GVNS. The parallel GVNS executes four tasks simultaneously. Each task, executed on one core, consists of applying the shaking procedure and the local search procedure one after another. Note that since shaking procedure generates a solution at random, the four shaking procedures, executed simultaneously, return four different solutions and therefore solutions returned by the local search procedure applied on each of these solutions are not necessarily the same. The best solution obtained after executing these four tasks is examined to be set to the new incumbent solution. If the change of incumbent solution occurs, the search process is resumed starting from the new incumbent solution. Whole process is repeated until reaching CPU time limit.

Computational results

The performances of the proposed approaches, GVNS algorithm (`Sequential GVNS`) and parallel GVNS algorithm (`Parallel GVNS`), have been disclosed on the benchmark instances provided by organizers of VeRolog 2014 solver challenge [218]. According to the size instances are classified as:

- small instances - about 50 customers and 15-20 swap locations;
- medium instances - about 200 customers and about 50 swap locations;
- large instances - about 500 customers and about 100 swap locations.

In addition for each instance size three types of instances may be distinguished:

- normal - there are customers that can be visited by train as well as those that must be visited only by truck;
- all_without_trailer - all customers must be serviced only by truck;

- all_with_trailer - all customers can be visited either by truck or train.

All tests have been carried out on a computer with Intel i7-4900MQ CPU 2.80 GHz and 16GB RAM. The CPU time limit has been set to 600 seconds as suggested by organizers of VeRolog 2014 solver challenge.

The computational results have been presented in the Table 2.7. For each GVNS version we provide the value of best solution found (Column `Value`) as well as time in seconds spent to reach that solution (Column `Time`). In column `dev` we report the percentage deviation of value found by sequential GVNS from corresponding value provided by parallel GVNS (which used four available cores).

	`Parallel GVNS`		`Sequential GVNS`		dev
Test instance	Value	Time(s)	Value	Time(s)	(%)
small_all_without_trailer	5249.18	1.96	5249.18	146.08	0.00
small_all_with_trailer	4731.02	6.08	4731.02	119.85	0.00
small_normal	4847.63	66.19	4847.63	187.28	0.00
medium_all_without_trailer	8382.80	464.59	8398.80	594.20	0.19
medium_all_with_trailer	7765.75	357.50	7754.39	594.58	-0.15
medium_normal	7834.78	497.37	7878.74	49.09	0.56
large_all_without_trailer	22310.60	430.16	22477.60	599.22	0.75
large_all_with_trailer	20066.40	465.37	20263.10	527.26	0.98
large_normal	20496.40	599.60	20570.2	591.81	0.36
preselection_large_all_without_trailer	26515.90	573.17	26631.70	550.12	0.44
preselection_large_all_with_trailer	24965.10	599.09	25272.30	580.85	1.23
preselection_large_normal	25443.20	583.27	25647.00	596.15	0.80
Average	14884.06	387.03	14976.81	428.04	0.43

Table 2.7: Computational results on benchmark instances

From the results presented in Table 2.7 we may infer that parallel GVNS is better option for solving SBVRP than sequential GVNS. Although sequential GVNS offered slightly better solution than parallel GVNS on medium_all_with_trailer instance on all the other instances parallel GVNS provided better solutions. Further, regarding average CPU time consumed by each of GVNS variant it follows that parallel GVNS is about 40 second faster than sequential GVNS, on the average. The success of parallel GVNS may be explained by the fact that it explores larger part of solution space than sequential GVNS within the same time limit. Namely, parallel GVNS at each iteration generates four solutions on which local searches are applied unlike sequential GVNS which performs local search on only one solution at each iteration.

2.4.5 Uncapacitated r-allocation p-hub median problem

The p-hub median problem consists of choosing p hub locations from a set of nodes with pairwise traffic demands in order to route the traffic between the origin-destination pairs at minimum cost [174]. Recently, Yaman [227] introduced a generalization of this problem in which each node can be connected to at most r of the p hubs, called the *Uncapacitated r-Allocation p-Hub Median Problem (r-p hub median problem)*. It is also a generalization of the multiple allocation p-hub median problem where each node can send and receive traffic through any of the p hubs [30, 31]. The motivation for this variant comes from the fact that the single allocation version is too restrictive, and the multiple allocation variant results in high fixed costs and complicated networks.

Specifically, the r-p hub median problem may be stated as follows. Given set N of n nodes and the distance matrix D, whose each entry d_{ij} represents the distance between nodes i and j. For every pair of nodes i and j, there is an amount of flow $t_{ij} \geq 0$ that needs to be transferred from i to j. It is generally assumed that transportation between non-hub nodes i,j is possible only via hub nodes h_i, h_j, to which nodes i,j are assigned, respectively. Transferring t_{ij} units of flow through path $i \to h_i \to h_j \to j$ induces a cost $c_{ij}(h_i, h_j)$, which is computed as

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

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

While the single and multiple allocation p-hub median problems have been widely studied in the literature [32, 68, 69, 70, 73, 133, 145, 146, 163], there are only two methods proposed for solving the r-p hub median problem. Besides, the exact method proposed in [227], Peiro et al. [180] proposed heuristic based on GRASP metaheuristic that employs three local search procedures and uses a mechanism to eliminate low-quality solutions during the greedy phase. Therefore, they selectively apply local searches to promising solutions.

For solving the r-p hub median problem, we propose a heuristic based on Variable Neighborhood Search (VNS) [167] that uses Nested Variable Neighborhood Descent (Nested VND) [133] as a local search. We continue the research direction established in [133] and [180]. In [180], three neighborhood structures are proposed, but explored within GRASP methodology. Additionally, neighborhoods were not explored in the nested manner. In [133], a nested VND variant was suggested in its general form, but it was not applied for solving the uncapacitated single allocation p-hub median problem with three neighborhood structures since it would be time consuming. Therefore, the authors proposed and tested two Mixed-nested VND procedures. (Note that [119] is the first paper where mixed-nested VND is suggested, i.e., in each point of the j-means neighborhood, h-means and k-means are applied sequentially.) We show that full nested VND, obtained by nesting two neighborhoods, is powerful enough, to enable us to efficiently and effectively solve r-p hub median problem (slightly different problem than the one considered in [133]). The merit of the proposed approach is disclosed on benchmark instances from the literature. It appears that the results obtained outperform those of the state-of-the-art heuristic based on GRASP. Moreover, the full Nested VND is used for the first time as a local search within VNS.

We represent a solution of the r-p hub median problem by a set H containing p hubs and a matrix A, where each row i contains r hubs assigned to node i (i.e., i-th row coincides with the set H_i). Thus, our solution is represented as $S = (H, A)$. The initial solution is constructed using the greedy heuristic [180]. Two neighborhood structures of a given solution $S = (H, A)$ are constructed. The first neighborhood structure, denoted by $\mathcal{N}_H$, is obtained by replacing one hub node from H by another non-hub node from $N \setminus H$. More formally,

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

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

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

Unfortunately, evaluating the value of each neighborhood solution from $\mathcal{N}_H$ requires solving allocation problem which is NP-hard [156]. So, solving exactly the associated allocation problem will be quite time consuming. Therefore, we find a near-optimal allocation A' of some solution $S'' \in \mathcal{N}_H$ as follows. For each node, we firstly determine the node-to-hub allocation using the greedy procedure (see Algorithm 35).

Algorithm 35: Greedy allocation

Function `GreedyAllocation(H, A)`;

for $i \in N$ **do**
  for $j = 1$ **to** p **do** $value(j) = d_{ih} \sum_{j \in N} t_{ij} + \sum_{j \in N} t_{ij} d_{hj}$;
  Sort array $value$ in nondecreasing order i.e.,
  $value(\pi(1)) \leq value(\pi(2)) \leq \cdots \leq value(\pi(p))$;
  for $j = 1$ **to** r **do** $A[i][j] = h_{\pi(j)}$

end

The solution $S'' = (H'', A'')$ obtained in this way is then improved by exploring the second neighborhood $\mathcal{N}_A(S'')$. In that way, the so-called nested variable neighborhood descent (Nested VND) is defined. The reason why we use the nested strategy for exploring both $\mathcal{N}_H$ and $\mathcal{N}_A$ neighborhood structures is the higher cardinality of the nested neighborhood $Nest(S)$:

$$|Nest(S)| = |\mathcal{N}_H(S)| \cdot |\mathcal{N}_A(S)|.$$

Such cardinality obviously increases chances to find an improvement in that nested neighborhood.

Computational results

All experiments have been carried out on a computer with Intel i7 2.8 GHz CPU and 16GB of RAM. The testing has been performed on two groups of instances available on the following address: `http://www.optsicom.es`:

- The AP (Australian Post) data set [68]. This set contains 231 instances with $n \in \{60, 65, 70, 75, 80, 85, 90, 95, 100, 150 \text{ and } 200\}$, p ranges from 3 to 8, while r takes values from 2 to $p - 1$, and $\gamma = 3, \alpha = 0.75, \delta = 2$.

These instances do not have a symmetric flows and flows from a node to itself can be positive.

- The USA423 data set [180]. From the original data with $n = 423$, 30 instances have been extracted, setting p equal to 3,4,5,6,7 and r to $2, 3, \ldots p-1$. For each combination of parameters p and r, the two different sets of the cost parameter values γ, α and δ are used: 0.1, 0.07, 0.09, and 0.09, 0.075, 0.08.

We tested GVNS which uses full nested VND as a local search, as well as its modification named GVNS_Reduced on both data sets. Namely, instead of performing complete exploration of the $\mathcal{N}_H(S)$, GVNS_Reduced examines just $m \cdot p$ of its elements. These elements were generated randomly, replacing each hub node with one of m randomly chosen non-hub nodes. The reason why we decided to perform such reduction relies on the fact that a neighborhood $\mathcal{N}_H(S)$ for instances in the second data set is much larger than the neighborhood $\mathcal{N}_H(S)$ for instances in the first data set. Therefore, exploring entirely neighborhood $\mathcal{N}_H(S)$ within GVNS applied on instances from the second data set might be time consuming. Note that m represents a parameter of GVNS_reduced algorithm, which defines the size of the reduced neighborhood. In order to possibly resolve local minima traps, both GVNSs use the shaking procedure that consists of replacing random k hubs of a given solution with random k non-hub nodes.

Comparisons with the state-of-the-art heuristic. We present comparisons of GVNS based heuristics with the GRASP heuristic. On AP data set GVNS and GVNS_Reduced are tested adjusting their parameters in the following way. The running time t_{max} is set to 60 seconds and 300 seconds for instances with $n < 80$ and with $n \geq 80$, respectively. On the second data set, for both GVNS variants time limit is set to 1h (3600 seconds). The parameter m of GVNS_Reduced is set to 10. The results obtained on both data sets are compared with those provided by the GRASP heuristic [180]. The GRASP heuristic were executed on a computer with characteristics similar to those of our computer (Intel i7 2.7GHz CPU and 4GB of RAM).

In Table 2.8, we provide comparison of solutions found by GVNS and

GVNS_Reduced with best solutions found by GRASP on instances from AP data set. The average value of best found solutions and average CPU times needed for finding these solutions over all instances with the same number of nodes are reported. The column headings are defined as follows. In the first column of Table 2.8, we report the number of nodes in the considered instances, whereas in the columns GRASP, GVNS and GVNS_Reduced, we provide the average of best solution values found by GRASP, GVNS and GVNS_Reduced, respectively. In columns Time, the average time needed to reach best found solutions for instances with n nodes are given, while in Column Impr.(%), we report the percentage improvement obtained by GVNS and GVNS_Reduced compared with the current best known values. From the reported results, it follows that within each set of instances with the same number of nodes, there is at least one instance where the best known solution is improved by GVNS and GVNS_Reduced. Moreover, the average improvement on AP data set achieved by GVNS variants is around 0.25%. It should be emphasized that just on one instance, the solution obtained by GRASP was better than the one reported by GVNS variants. Comparing average CPU times needed to solve instances, we conclude that GVNS_Reduced outperforms both GVNS and GRASP.

	GRASP		GVNS			GVNS_Reduced		
	aver.	aver.	aver.	aver.	%	aver.	average	%
n	value	time	value	time	impr.	value	time	impr.
60	122348.90	4.59	121829.27	3.73	0.42	121829.27	3.46	0.42
65	123001.53	6.66	122689.74	5.87	0.25	122689.74	8.27	0.25
70	123931.76	10.51	123604.38	5.75	0.26	123604.38	8.64	0.26
75	124776.42	11.11	124650.73	5.93	0.10	124650.73	7.52	0.10
80	125148.22	14.40	124844.76	9.36	0.24	124844.76	10.23	0.24
85	125566.58	19.48	125378.23	13.10	0.15	125378.23	15.79	0.15
90	124934.99	22.95	124734.55	12.32	0.16	124744.00	11.52	0.15
95	125121.18	24.27	124926.55	25.45	0.16	124931.36	19.35	0.15
100	125805.04	4.81	125588.19	10.39	0.17	125588.19	9.73	0.17
150	126728.85	21.42	126307.10	24.70	0.33	126310.54	29.08	0.33
200	129144.44	58.86	128788.66	98.67	0.28	128913.20	41.83	0.18
Avg.	125137.08	18.10	124849.29	19.57	0.23	124862.22	15.04	0.22

Table 2.8: Comparison of GRASP and GVNS on AP instances

In Table 2.9, we report summarized results on USA423 data set. On

these instances, GVNS and GVNS_Reduced have been tested setting their parameters in the previously described way. The obtained results are compared with the results obtained by GRASP [180]. For each choice of parameters γ, α, δ, and for each method, we report the average of best found solution values (columns 'av.value'), and average CPU times (columns 'av.time') consumed to reach best found solutions for all instances with the same parameter values. In columns ' impr.(%)', we report the average percentage improvement of solution values achieved by GVNS variants relatively to GRASP. From the reported results, we conclude that both GVNS variants outperform GRASP heuristic regarding both solution quality and CPU time. Also, it is noted that GVNS_Reduced is significantly faster than GVNS. That could be explained by the fact that nested VND employed within GVNS_Reduced has smaller complexity than the one used inside GVNS. Further, regarding average solution improvement, GVNS_Reduced performs better than GVNS on the instances whose parameters γ, α, δ are set to 0.1, 0.07, 0.09 respectively. On the other hand, GVNS achieves greater average improvement on the instances whose parameters γ, α, δ take values 0.09, 0.075 and 0.08, respectively. Moreover, for each tested instance, either GVNS or GVNS_Reduced provided new best known solution. Note that the average results obtained by GVNS_Reduced outperform GVNS on the instances where parameters γ, α, δ are set to 0.1, 0.07, and 0.09 respectively, despite the fact that GVNS_Reduced does not perform complete exploration of the $\mathcal{N}_H(S)$, which GVNS does. This can be explained by the fact that the stopping condition is represented by maximum CPU time allowed for the search. Therefore, sometimes it is more beneficial to explore the subset of the neighborhood than the whole neighborhood, reducing the CPU time for the intensification but increasing the time for diversification (see e.g., [121]).

parameters			GRASP		GVNS			GVNS_Reduced		
γ	α	δ	aver. value	aver. time	aver. value	aver. time	% impr.	aver. av.value	aver. time	% impr.
0.1	0.07	0.09	28673188808	2880	28541499618	1115	0.46	28474980301	682	0.70
0.09	0.075	0.08	25767289897	2880	25552243305	1288	0.84	25610842214	690	0.61

Table 2.9: Comparison of GRASP and GVNS variants on USA423 instances

Finally, we use two well-known nonparametric tests for pairwise comparisons: Wilcoxon test and the Sign test to compare the proposed GVNS variants against the GRASP heuristic, taking into account all considered test instances. Wilcoxon test enables us to get the answer on the question whether two samples represent two different populations or not, while the Sign test computes the number of instances on which an algorithm improves upon the other algorithm. p-values of 0.000, returned by both tests, when used to compare GVNS with the GRASP, and GVNS_Reduced with GRASP, confirm the superiority of GVNS variants over the GRASP heuristic.

2.4.6 Minimum sum-of-squares clustering on networks

The aim of Cluster Analysis is to partition a set of entities into clusters, so that entities within the same cluster are similar and entities in different clusters are different [116]. The most well-known model is *Minimum Sum-of-Squares Clustering* (MSSC). In its basic form, MSSC assumes the entities to be points in $\mathbb{R}^n$; p points (called *prototypes*) are sought by minimizing the sum of the squares of the Euclidean distances separating the entities from their closest prototypes. MSSC, recently shown to be NP-hard in [3], can be solved exactly for data sets of moderate sizes (over 2,300 entities) by column generation [4]. For larger data sets, heuristics are used, see [119, 123] and the references therein.

We assume that the set V of entities to be clustered is the set of nodes of a connected and undirected network $G = (V, E)$, where E is a collection of pairs of nodes, and each $e \in E$ has positive length l_e. The set V of nodes is to be split into p disjoint subsets (clusters, communities), and p prototypes (one per cluster) are to be chosen so that all entities in a given cluster are close to their associated prototype. We propose to use a sum-of-squared-distance criterion, which leads us to address two versions of the problem, called V-MSSC and E-MSSC. The V-MSSC (*vertex*-MSSC) consists of selecting a subset V^* of p entities (prototypes) within the set of vertices of the network, so that the sum of squares of the distances from each entity $v \in V$ to its closest prototype is minimized. Closeness between any pair of entities $u, v \in V$ is measured by the length $d(u, v)$ of the shortest path connecting u and v. The E-MSSC

(*edge*-MSSC), [34], has the same objective, but the prototypes are sought not only at vertices, but also at points along the edges of the network. This way, one may obtain with the E-MSSC clusters configurations which are impossible if only vertices are allowed to be prototypes. In other words, the clusters class which can be obtained by solving E-MSSC is richer than the one obtained by solving V-MSSC problems, and this may lead to more accurate clusters. It is interesting to note that the difference between vertex minimum sum-of-distances and edge minimum sum-of-distances (not squared distances) does not exist, since the optimal solutions of both are equivalent [108].

Structural properties: We present now some properties of the E-MSSC, extending the results given in [34]. These will be helpful in the design of the Basic VNS heuristic algorithm described hereafter. Whereas prototypes are allowed to be located at the interior of edges, they cannot be concentrated on a given edge, since each edge can contain at most one optimal prototype in its interior, as stated in the following.

Proposition 2.4.3 *Let $(x_1^*, \ldots, x_p^*)$ be an optimal solution to E-MSSC and let $e = (u, v) \in E$. The interior of e contains at most one optimal prototype. If it contains one optimal prototype x_j^*, then both endpoints u and v have x_j^* as the closest prototype.*

By Proposition 2.4.3, given a node v, if an edge e adjacent to v contains in its interior some optimal prototype x_j^*, then both endpoints of e, including v, must have x_j^* as their closest prototype, and thus v cannot be an optimal prototype. This implies the following

Proposition 2.4.4 *If an optimal prototype is located at a node $v \in V$, then the interior of all edges adjacent to v contains no optimal prototypes.*

E-MSSC is a nonlinear optimization problem defined on a network. However, the optimal solution of E-MSSC in the case $p = 1$ may be easily determined. Namely, for $p = 1$, it is easy to construct a finite dominating set [131], i.e., a set known to contain an optimal solution. To construct such finite dominating set, it is important to recall that, given $v \in V$, the distance

from v to a point x in a given edge $e \in E$ with endpoints u_1 and u_2 is given by

$$\begin{aligned} d(v,x) &= \min\{d(v,u_1) + d(u_1,x), d(v,u_2) + d(u_2,x)\} \\ &= \min\{d(v,u_1) + d(u_1,x), d(v,u_2) + l_e - d(u_1,x)\}. \end{aligned} \tag{2.9}$$

Whether the minimum above is attained at the first or the second term depends on the relative position of point x with respect to the so-called *bottleneck point*, [147], $z(v)$, defined as

$$z(v) = \frac{1}{2}\left(l_e - d(v,u_1) + d(v,u_2)\right). \tag{2.10}$$

Formula (2.10) allows us to compute the distance from node v to any point in the edge e :

- If $z(v) \leq 0$, then the shortest path from v to any point $x \in e$ passes through u_2.
- If $z(v) \geq l_e$, then the shortest path from v to any point $x \in e$ passes through u_1.
- If $0 < z(v) < l_e$, then the shortest path from v to $x \in e$ passes through u_1 if x belongs to the sub-edge with endpoints u_1 and $z(v)$, and it passes through u_2 if x belongs to the sub-edge with endpoints $z(v)$ and u_2.

In the latter case, such $z(v)$ will be called a bottleneck point. By definition, for such v, the distance from v to $z(v)$ via u_1 is equal to the distance from v to $z(v)$ via u_2, and then two shortest-paths exist from v to $z(v)$.

Given an edge $e \in E$, and $v \in V$ the distance to v is an affine function two subintervals (possibly degenerate) in which the bottleneck $z(v)$ splits e. Hence, within each such subinterval, the squared distance is a quadratic polynomial in the variable x. This process can be done on any given edge e for all nodes v, calculating all bottleneck points $z(v)$, and splitting e into $O(|V|)$ subintervals, such that, within each subinterval, each squared distance is a quadratic polynomial function, and thus the sum of the squared distances is also a quadratic polynomial function. In other words, the objective function of

E-MSSC is a second-degree polynomial function in one variable on each such interval. Thus, the derivative of its minimum point should be equal to zero, if it belongs to the considered interval; otherwise its minimum is achieved at one of endpoints of the considered interval. Therefore, at each edge e, the objective function can have at most $|V|-1$ local minima, obtained analytically. By doing this process for all edges, we obtain a finite dominating set D for E-MSSC. Since the optimal point has to belong to the finite dominating set D, it can easily be obtained by its inspection.

Variable Neighborhood Search for solving E-MSSC: In the case $p > 1$, finding a globally optimal solution may be done by inspecting, for all possible partitions $C_1, \ldots, C_p$ of V, the objective value at $(x_1^*, \ldots, x_p^*)$, where each x_j^* is the optimal solution to E-MSSC for $p = 1$. However, this approach is only applicable for networks of very small size. Hence, for large data sets, we propose Basic VNS. Before, providing more details about Basic VNS we describe the solution space of E-MSSC and neighborhood structures of a solution. Feasible solutions of E-MSSC are identified with sets $x = \{x_j : x_j \in e \text{ with } e \in E\}$ of cardinality p. A (symmetric) distance function ρ can be defined on the set of solutions as

$$\rho(x, x') = |x \setminus x'| = |x' \setminus x| \qquad \forall x, x'.$$

So, the distance between two solutions x, x' is equal to k if and only if the sets x and x' differ exactly in k locations. The neighborhood structure $\mathcal{N}_k(x)$, $k \leq p$, of a solution x, induced by the metric ρ, is defined as the set of all solutions x' deduced from x by replacing k locations from x with k locations that are not in x.

The proposed Basic VNS, denoted by `Net-VNS`, as an initial solution takes either a random solution or a more sophisticated solution described hereafter. The random solution is created by randomly selecting points which do not violate the aforementioned structural properties. Given a k-uple ($k < p$) of prototypes already selected, we say that a point of the network is feasible if, together with those prototypes previously chosen, satisfies Propositions 2.4.3 and 2.4.4. We say an edge is feasible if it contains feasible points. In our

randomized procedure, at each step one feasible edge is chosen at random, and then one feasible point in such edge is chosen at random. The process is repeated until p points are obtained.

As a local search `Net-VNS` uses a location-allocation heuristic, Algorithm 36, in which we exploit the fact that, in the location step, the p independent subproblems to be solved are easy, since, as discussed above, they can be solved by inspection of the low-cardinality set of candidate points (i.e., the subproblem is actually E-MSSC with $p = 1$).

Algorithm 36: K-Net-Means algorithm (`Net-KM`) for the NMSSC problem

Function `NetKM` (n, p, x);
1 $C_j = \emptyset, j = 1, \ldots, p$;
2 **repeat**
3 **for** $v \in V$ **do**
4 $m_v \leftarrow \arg\min_{x_j \in x} d^2(v, x_j)$; $//m_v \in \{1, \ldots, p\}$
5 $C_{m_v} = C_{m_v} \cup \{v\}$;
  end
6 `RemoveDeg`(n, p, C) ;
7 **for** $j := 1, \ldots, p$ **do**
8 calculate prototype x_j;
  end
until *stopping criterion is satisfied*;

Starting from a set of p initial prototypes (e.g., taken at random), users are assigned to their closest prototype (steps 3-5). Each user v is assigned to the cluster C_{m_v}, where m_v is the index of a prototype closest to v. In the case of ties, i.e., if there are more than one prototype with the same distance to v, the one with the smallest index is chosen. Steps 7 and 8 are location steps, where prototypes x_j, $j = 1, \ldots, p$ are found for a given clusters C_j. More precisely, for each cluster a 1-prototype problem is solved. Allocation step of `NetKM` is repeated with the new locations of the prototypes. These steps are repeated until no more changes in assignments occur.

When using this local-search procedure, we may face *degeneracy* problems [23, 35]: when customers are allocated to prototypes, some prototypes may

remain with no customers assigned. Obviously, if we move one of such prototypes to any node, we will strictly improve the objective value. So, if a degenerate configuration is obtained during this local-search procedure, all prototypes without nodes allocated can be moved randomly to remaining nodes that are not used already as a prototype, improving the objective value. Thus, the statement `RemoveDeg`(n, p, C) in the algorithm `NetKM` refers to resolving the degeneracy issue (if it is detected) in the previously described way.

The shaking procedure used within `Net-VNS` at the input requires a solution x and the parameter k, while at the output it returns a random solution from the neighborhood $\mathcal{N}_k(x)$.

Computational results

The aim of this section is to explore whether the new model, E-MSSC, is essentially different from the V-MSSC, by checking if the clusters obtained are the same or not to those given when only entities (nodes) are allowed to be prototypes. For solving V-MSSC we use VNS based heuristics described in [118], which we denote with `VNS-0`. For solving E-MSSC we apply three different VNS based heuristic, `VNS-1`, `VNS-2` and `VNS-3`. `VNS-0` heuristic follows rules of Basic VNS. It explores neighborhood structures induced by metric ρ using Interchange (or vertex substitution) heuristic as a local search. As an initial solution, p nodes are selected at random. The initial solution of `VNS-1` is obtained by `VNS-0`. Then, the local-search described in Algorithm 36 is performed. Algorithm `VNS-2` uses as a starting solution one obtained by `VNS-1`, and then our `Net-VNS` is run. Finally, `VNS-3` starts with the random initial solution, followed by our `Net-VNS`.

All algorithms, described in the previous paragraph, have been tested on 40 p-median instances taken from the OR-Lib [15]. Each algorithm has been run on each instance 10 times, for the different choice of a seed, with the time limit of 10 seconds. The results, obtained with a personal computer with a 2.53 GHz CPU and 3 GB of RAM, are reported in Table 2.10. The column headings are defined as follows. The first column, `Instance`, gives the name of the OR-Lib instances. Instances parameters, namely, the number

n of entities, and the number p of prototypes sought, are given in columns 2 and 3 respectively. Columns 4 and 5 report the results of two different heuristics: in column `V-MS` we have chosen as prototypes those p entities minimizing the sum of distances from the entities to the closest prototype, i.e., the optimal solution to the p-median problem, while in column `V-MSS` we report the value of the best solution of V-MSSC obtained during ten runs. In both cases, the problem is not solved exactly, but using the version of the VNS for solving p-median described in [118], i.e., `VNS-0`. The best objective values $f_{\mathtt{V-MS}}$ and $f_{\mathtt{V-MSS}}$ of the prototypes obtained when solving both problems are compared: column `dev.` reports the % deviation between these two values (i.e., dev(`V-MS`,`V-MSS`)). The next column reports the best solution value (column `E-MSSC`) obtained by one of three different variants of VNS applied to the E-MSSC problem (`VNS-1, VNS-2, VNS-3`) in ten runs. The deviation of this value from the value reported in column `V-MSS` is reported in column 8. It should be emphasized that almost all values reported in column `E-MSSC` were found by `VNS-2`. The only exception is instance `pmed2`, on which `VNS-3` found the best value. Finally, the last two columns analyze whether E-MSSC yields solutions which are not obtained when only entities are considered as prototypes. Column `Node` indicates whether the set of optimal prototypes, which corresponds to the value reported in column `E-MSSC` only contains nodes (and coincides with the optimal prototypes for the V-MSSC problem). Even of the set of optimal prototypes of E-MSSC and V-MSSC do not coincide, it may be the case that they yield identical clusters. This is reported in the last column, `Same`.

From Table 2.10 the following observations may be derived.

1. Comparing values reported in columns `V-MSS` and `E-MSS`, one can observe that the difference between these values mostly depend on the value of p. For almost all instances with $p = 5$ or 10, especially those with large n, the best obtained values for V-MSSC and E-MSSC are the same, as well as yielded clusters. On the other hand, larger values of p implies the significant lower value for E-MSSC and the different clusters than those obtained by V-MSSC.

2. In 12 out of 40 instances, the prototypes of E-MSSC and V-MSSC

Instance	n	p	V-MS	V-MSS	dev. (%)	E-MSS	dev. (%)	Node	Same
pmed1	100	5	450233	450233	0.00	450043.94	0.04	no	no
pmed2	100	10	271829	256874	5.50	253067.60	1.48	no	no
pmed3	100	10	295752	263385	10.94	259643.17	1.42	no	no
pmed4	100	20	159678	153963	3.58	147685.50	4.08	no	no
pmed5	100	33	45055	42671	5.29	40066.36	6.10	no	no
pmed6	200	5	410360	406195	1.01	386642.24	4.81	no	no
pmed7	200	10	222901	221631	0.57	221602.83	0.01	no	yes
pmed8	200	20	157807	151558	3.96	151094.71	0.31	no	no
pmed9	200	40	68886	66525	3.43	63126.34	5.11	no	no
pmed10	200	67	16199	15938	1.61	14917.01	6.41	no	no
pmed11	300	5	256532	256532	0.00	256512.74	0.01	no	no
pmed12	300	10	197970	197814	0.08	197814.00	0.00	yes	yes
pmed13	300	30	100398	99210	1.18	98471.40	0.74	no	yes
pmed14	300	60	53604	49977	6.77	49152.57	1.65	no	no
pmed15	300	100	20593	20213	1.85	18653.68	7.71	no	yes
pmed16	400	5	210452	209886	0.27	209886.00	0.00	yes	yes
pmed17	400	10	160401	160401	0.00	160401.00	0.00	yes	yes
pmed18	400	40	92325	88234	4.43	87499.01	0.83	no	no
pmed19	400	80	35678	33782	5.31	32292.46	4.41	no	no
pmed20	400	133	16769	16032	4.40	14930.55	6.87	no	no
pmed21	500	5	203552	203552	0.00	203552.00	0.00	yes	yes
pmed22	500	10	189091	188857	0.12	188857.00	0.00	yes	yes
pmed23	500	50	67359	66257	1.64	65834.72	0.64	no	no
pmed24	500	100	30715	29478	4.03	28533.80	3.20	no	no
pmed25	500	167	13736	13377	2.61	12502.12	6.54	no	no
pmed26	600	5	199503	199503	0.00	199503.00	0.00	yes	yes
pmed27	600	10	147401	147096	0.21	147096.00	0.00	yes	yes
pmed28	600	60	52546	51239	2.49	51030.45	0.41	no	yes
pmed29	600	120	27143	25848	4.77	25335.36	1.98	no	no
pmed30	600	200	12755	12533	1.74	11671.78	6.87	no	no
pmed31	700	5	172938	171963	0.56	171963.00	0.00	yes	yes
pmed32	700	10	157283	157177	0.07	157177.00	0.00	yes	yes
pmed33	700	70	49432	47255	4.40	47188.77	0.14	no	yes
pmed34	700	140	22807	21981	3.62	21461.21	2.36	no	yes
pmed35	800	5	160564	160564	0.00	160541.91	0.01	no	no
pmed36	800	10	153164	152914	0.16	152914.00	0.00	yes	yes
pmed37	800	80	50665	48246	4.77	48195.16	0.11	no	yes
pmed38	900	5	161102	161102	0.00	161102.00	0.00	yes	yes
pmed39	900	10	126553	125175	1.09	125175.00	0.00	yes	yes
pmed40	900	90	44596	43035	3.50	42877.82	0.37	no	no

Table 2.10: Computational results

coincide (see column `Node`). On the other hand, in 19 instances out of 40 (47.5%, see column `Same`), the clusters obtained by the two methods are the same. This ratio is much lower than the one reported by the authors in [34], whose preliminary results on clustering on the line

showed that 80% of the cases considered gave the same partitions.

3. The classical p-median problem usually yields very good solutions for V-MSSC, as seen when comparing columns `V-MS` and `V-MSS`, so they could be used almost indifferently to build the starting solution of `VNS-2`.

2.4.7 Periodic maintenance problem

The *periodic maintenance problem* (PMP) [107] is stated in the following way. There is a set of machines $M = \{1, 2, \ldots, m\}$, and there is a set of periods $U = \{1, 2, \ldots, T\}$ with $T \geq m$. The PMP consists of finding an optimal cyclic maintenance schedule of length T that is indefinitely repeated. At most one machine is serviced at each period and all the machines must be serviced at least once for any cycle. When machine $i \in M$ is serviced, a given non-negative servicing cost of b_i is incurred, regardless of the period. At period $t \in U$, a machine $i \in M$ that is not serviced during some period is in operation and incurs an operation cost of $n_i(t) \times a_i$ where a_i is a given positive integer number, and where $n_i(t)$ is the number of periods elapsed since last servicing of machine i. The main objective of this problem is to determine a feasible maintenance schedule with a minimum cost, i.e., to decide for each period $t \leq T$ which machine to service (if any), so that the total servicing costs and operating costs are minimized. Note that if cycle length T is a decision variable then the problem is called the *Free periodic maintenance problem*. However, here we consider T as an input parameter.

Based on the problem definition (including $T \geq m$), the features of test instances (described hereafter) and problem characteristics, it may be concluded that the solution space of PMP consists of all vectors $\pi = (\pi_1, \pi_2, \ldots, \pi_T)$, with $\pi_t \in M$ for $t \in U$ such that $M \subset \pi$. In such representation, π_t corresponds to the index of a machine serviced in the t^{th} time period. In what follows the solution space of PMP will be denoted by P.

In order to explore the solution space we propose new variant of VNS that we call Nested GVNS (NGVNS). It may be seen as an extension of the nested variable neighborhood descent (Nested VND) already proposed in [126, 133]. It applies general variable neighborhood search (GVNS) on each element of

the preselected neighborhood structure, unlike Nested VND that applies a sequential Variable neighborhood descent instead.

The proposed NGVNS applies GVNS starting from each element of the neighborhood structure *Replace* of the current solution π. The neighborhood *Replace*(π) contains all sets $\pi' \in P$ that may be derived from the set π by replacing one element of π (say π_j) with one from the set M, e.g., $\pi_k \in M,\ \pi_k \neq \pi_j$.

If an improvement is detected, it is accepted as a new incumbent solution π and whole process is repeated starting from that solution. NGVNS finishes its work if there is no improvement. An initial solution for the proposed NGVNS is built as follows. In the first m periods all m machines are chosen to be serviced. In the remaining $T - m$ periods, machines to be serviced are chosen at random. In that way the feasibility of the initial solution is achieved. The reason why we decide to use NGVNS instead of Nested VND is that solution obtained by GVNS can not be worse than one obtained by VND, used within GVNS. Therefore the solution quality found by NGVNS is at least as good as one found by Nested VND.

The GVNS used within NGVNS includes a shaking phase used in order to escape from the local minima traps and an intensification phase in which sequential VND (seqVND) is applied. Within seqVND the following neighborhood structures of a solution π are explored:

- $Reverse\_two\_consecutive(\pi)$ (1-opt) - the neighborhood structure consists of all solutions obtained from the solution π swapping two consecutive elements of π.

- $Shift\_backward(\pi)$ (Or-opt) - the neighborhood structure consists of all solutions obtained from the solution π moving some element π_t backward immediately after some element π_s for all $s < t$.

- $Shift\_forward(\pi)$ (Or-opt)- the neighborhood structure consists of all solutions obtained from the solution π moving some element π_t immediately after some element π_s for all $s > t$.

- $Reverse\_part(\pi)$ (2-opt) - the neighborhood structure consists of all solutions obtained from the solution π reversing a sub-sequence of

π. Each solution in this neighborhood structure is deduced from the solution π reversing the part starting at π_t and ending at π_s $(t < s)$.

The Shaking phase of GVNS takes as the input the solution π and the parameter k. At the output it returns the solution obtained after performing k-times random shift move on π. Each random shift consists of inserting an element in π at random either backward or forward ($Shift\_backward$ and $Shift\_forward$).

Computational results

We compare five methods for solving the PMP. Four among them are exact and differ in mathematical programming formulation: our new formulation [209] and another three taken from [107]. The last method included in comparison is NGVNS based heuristic. The numerical experiments were carried on a personal computer with 2.53GHz CPU and 3GB RAM memory. All mathematical models, except set-partitioning (SP) model, were solved using the MIP solver IBM ILOG CPLEX 12.4. The time limit for MIP solver were set to 300 seconds.

For testing purposes we consider the same test instances proposed in Grigoriev et al. [107]. Comparative results are reported in tables 2.11-2.15. All results reported in tables 2.11-2.15 are obtained on the same computer except those obtained by SP based model. We simply copied them from [107]. In the tables the following abbreviations are used:

- `OPT` - optimal solution value.
- `MIP value`- value of solution obtained solving MIP formulation proposed in Grigoriev et al. [107].
- `MIP time`- CPU time, in seconds, consumed by MIP solver to solve MIP formulation proposed in Grigoriev et al. [107].
- `SP` - consumed CPU time, in seconds, for solving an instance using set-partitioning (SP) formulation.

- `FF` - CPU time, in seconds, needed to solve an instance using flow formulation (FF) formulation.
- `new-MIP` - CPU time, in seconds, spent by MIP solver to solve new MIP formulation proposed by us [209].
- `NGVNS` - CPU time (in seconds) consumed by NGVNS based heuristic to solve an instance of PMP.

T	a	OPT	MIP (s)	SP (s)	new-MIP (s)	FF (s)	NGVNS (s)
3	1 1 1	3	0.11	1	0.01	0.06	0.00
3	2 1 1	4	0.02	1	0.02	0.03	0.00
3	2 2 1	5	0.02	1	0.02	0.03	0.00
4	5 1 1	5.5	0.05	1	0.03	0.06	0.00
4	5 2 1	7	0.03	1	0.02	0.03	0.00
5	5 5 1	10	0.04	1	0.19	0.02	0.00
4	10 1 1	8	0.02	1	0.02	0.02	0.00
4	10 2 1	9.5	0.03	1	0.02	0.00	0.00
6	10 5 1	13.3333	0.04	1	0.05	0.05	0.01
16	10 10 1	17.25	3.04	114	0.16	0.61	0.00
8	30 1 1	14.5	0.10	1	0.08	0.03	0.00
17	30 2 1	17.2941	3.32	89	0.55	0.69	0.10
8	30 5 1	22.25	0.07	1	0.11	0.03	0.02
9	30 10 1	28.4444	0.05	1	0.09	0.11	0.02
13	30 30 1	42.9231	0.16	9	0.11	0.03	0.09
10	50 1 1	19	0.09	1	0.08	0.03	0.02
21	50 2 1	22.6667	11.74	604	1.64	0.98	0.12
10	50 5 1	29.5	0.15	2	0.08	0.05	0.03
10	50 10 1	36.5	0.14	1	0.09	0.03	0.03
15	50 30 1	55	0.37	27	0.17	0.73	0.10
17	50 50 1	66.8235	0.57	114	0.16	0.62	0.05
	Average time		0.960	46.333	0.176	0.202	0.028

Table 2.11: Instances with three machines ($m = 3, b_i = 0, i \in M$)

Note that Table 2.14 does not provide results obtained by SP model since not all solution values had been provided in [107].

The Table 2.16 provides the average CPU times consumed by each solution approach on a considered data set.

From Tables 2.11-2.16 the following conclusions may be drawn:

T	a	OPT	MIP value	MIP time(s)	SP (s)	new-MIP (s)	FF (s)	NGVNS (s)
4	1 1 1 1	6	6	0.04	1	0.05	0.00	0.00
9	2 1 1 1	7.3333	7.3333	0.35	1	0.16	0.05	0.02
10	2 2 1 1	8.8	8.8	0.68	1	0.14	0.05	0.05
15	2 2 2 1	10.4	10.4	8.84	1	0.34	0.72	0.04
6	5 1 1 1	10	10	0.06	1	0.11	0.02	0.00
16	5 2 1 1	11.75	11.75	17.84	1	0.33	0.78	0.05
22	5 2 2 1	13.7273	13.7273	300.01	3	1.12	2.32	0.21
6	5 5 1 1	15	15	0.07	1	0.09	0.03	0.00
6	5 5 2 1	17.5	17.5	0.06	1	0.09	0.02	0.00
24	5 5 5 1	22.25	22.25	300.16	3	2.45	2.25	0.02
6	10 1 1 1	12.5	12.5	0.08	1	0.05	0.01	0.00
6	10 2 1 1	15	15	0.07	1	0.09	0.01	0.00
6	10 2 2 1	17.5	17.5	0.09	1	0.09	0.02	0.00
8	10 5 1 1	19.5	19.5	0.27	1	0.12	0.09	0.01
6	10 5 2 1	22.5	22.5	0.08	1	0.06	0.03	0.00
8	10 5 5 1	27.875	27.875	0.16	1	0.16	0.22	0.00
8	10 10 1 1	24.5	24.5	0.13	1	0.12	0.02	0.01
6	10 10 2 1	27.5	27.5	0.10	1	0.11	0.01	0.00
9	10 10 5 1	34	34	0.26	1	0.14	0.08	0.00
33	10 10 10 1	40.4545	**41.0606**	300.02	17	5.04	3.62	0.89
8	30 1 1 1	21.75	21.75	0.18	1	0.11	0.03	0.00
8	30 5 1 1	29.5	29.5	0.15	1	0.09	0.06	0.01
10	30 5 5 1	40.5	40.5	0.42	1	0.11	0.10	0.02
8	30 10 1 1	37	37	0.13	1	0.08	0.05	0.03
12	30 10 5 1	49.6667	49.6667	2.09	1	0.17	0.18	0.04
30	30 10 10 1	58.3333	58.3333	300.18	19	6.75	4.52	0.66
26	30 30 1 1	55.8462	55.8462	300.01	3	13.81	2.36	0.42
24	30 30 5 1	70.5	70.5	300.01	4	3.18	3.20	0.35
14	30 30 10 1	81.5	81.5	3.02	1	0.36	1.25	0.05
19	30 30 30 1	108.4737	108.4737	21.09	1	0.69	1.17	0.22
		Average time		61.890	2.433	1.207	0.776	0.103

Table 2.12: Instances with three machines ($m = 3, b_i = 0, i \in M$)

i) Overall NGVNS approach appears to be most reliable. It solved all test instances to the optimality in the shortest CPU time (i.e., 0.831 seconds on average for all test instances).

ii) NGVNS needed in most of instances less than a second to get an optimal solution, except on instances in Table 2.15 which appear to be the hardest.

Regarding exact solution methods, several interesting observations may be

T	OPT	MIP value	MIP time(s)	SP (s)	new-MIP (s)	FF (s)	NGVNS (s)
50	3.04	3.04	300.02	51	3.81	4.52	0.75
51	3	3	25.96	21	0.47	3.54	1.84
52	3.0385	3.0385	300.01	154	4.77	5.40	0.38
53	3.0377	3.0377	107.24	247	3.20	4.98	0.38
54	3	3	219.56	32	0.76	4.76	1.66
55	3.0364	3.0364	53.13	117	4.26	5.63	0.68
56	3.0357	3.0357	300.12	590	4.73	5.13	0.46
57	3	3	16.37	46	0.42	3.95	1.43
58	3.0345	3.0345	300.01	366	5.63	4.99	2.42
59	3.0339	3.0339	164.97	407	5.12	4.68	1.41
60	3	**3.0667**	300.06	170	1.78	6.50	1.51
61	3.0328	3.0328	300.01	715	5.97	6.27	2.47
62	3.0323	**3.0882**	300.08	1437	4.79	7.67	1.47
63	3	3	233.27	195	1.04	6.04	2.93
64	3.0313	**3.0571**	300.08	1431	7.77	6.85	3.28
65	3.0308	**3.05**	207.69	1098	6.15	7.93	1.70
66	3	**3.0444**	46.11	470	2.62	5.57	2.81
67	3.0299	**3.16**	10.51	546	8.52	6.26	1.18
68	3.0294	**3.0882**	300.09	783	5.76	7.69	3.34
69	3	3	229.87	601	3.46	7.91	0.88
70	3.0286	**3.0571**	300.06	5150	6.52	7.53	1.87
80	3.025	3.025	300.17	1167	10.84	16.07	13.76
90	3	**3.0444**	300.06	1093	3.87	17.82	7.16
100	3.02	3.02	300.11	2618	16.30	24.85	16.11
	Average time		217.315	812.708	4.940	7.606	2.994

Table 2.13: Instances with three machines ($m = 3, a_i = 1, b_i = 0, i \in M$)

derived:

i) The best performance on average is exhibited by FF and our new MIP formulation. Further, the optimality of found solutions for 4 test instances (Table 2.15) were not proven solving new MIP formulation, while solving FF formulation the optimality were not proven for 6 test instances (3 instances in Table 2.14 and 3 instances in Table 2.15). For all these instances reported times are boldfaced.

ii) The advantage of FF over new MIP formulation comes from the fourth test instances in Table 2.15. There, the optimal solution was reached by new MIP formulation but not proven in 300 seconds.

iii) The behavior of SP formulation is interesting. It is the worst exact

a	b	OPT	MIP value	MIP time(s)	new-MIP (s)	FF (s)	NGVNS (s)
5 1 1 1 1	0 0 0 0 0	15	15	300.01	2.48	2.08	0.22
5 1 1 1 1	5 1 1 1 1	17.3333	17.3333	300.01	3.00	2.26	0.36
5 1 1 1 1	30 10 5 2 1	27.0417	**27.125**	300.00	6.49	8.94	0.58
5 5 1 1 1	0 0 0 0 0	21.9583	21.9583	300.01	31.84	2.51	0.32
5 5 1 1 1	5 5 1 1 1	25.4167	25.4167	300.00	57.49	25.96	0.04
5 5 1 1 1	30 10 5 2 1	33.8333	**34.125**	300.00	9.42	28.28	0.61
5 5 5 1 1	0 0 0 0 0	29.5	29.5	300.15	5.21	2.78	0.13
5 5 5 1 1	5 5 5 1 1	33.5	**33.9167**	300.01	4.63	5.46	0.06
5 5 5 1 1	30 10 5 2 1	41.125	41.125	300.00	6.94	39.61	0.33
5 5 5 5 1	0 0 0 0 0	40.375	40.375	300.14	229.15	8.72	0.66
5 5 5 5 1	5 5 5 5 1	44.875	44.875	300.02	111.71	6.16	0.04
5 5 5 5 1	30 10 5 2 1	50.375	50.375	300.17	86.30	**300.00**	0.33
10 5 1 1 1	0 0 0 0 0	26.75	26.75	300.07	9.89	2.95	0.48
10 5 1 1 1	10 5 1 1 1	32.125	**32.25**	300.12	6.71	59.33	0.35
10 5 1 1 1	30 10 5 2 1	41	41	300.17	27.88	38.39	0.3
10 10 5 1 1	0 0 0 0 0	43.5	**43.9167**	300.03	48.75	4.65	0.6
10 10 5 1 1	10 10 5 1 1	50.9583	50.9583	300.01	49.17	6.86	0.3
10 10 5 1 1	30 10 5 2 1	56.125	**56.625**	300.10	21.19	**300.00**	0.87
30 10 5 1 1	0 0 0 0 0	61.4167	61.4167	300.11	30.5	4.18	0.28
30 10 5 1 1	30 10 5 1 1	77.4167	77.4167	300.10	38.14	13.53	0.62
30 10 5 1 1	30 10 5 2 1	77.5	**77.7083**	300.10	43.34	208.28	0.52
30 30 1 1 1	0 0 0 0 0	69	69	300.09	55.72	3.26	0.46
30 30 1 1 1	30 30 1 1 1	91.75	91.75	300.06	26.40	3.49	0.22
30 30 1 1 1	30 10 5 2 1	84.6667	84.6667	300.08	79.14	261.57	0.55
30 30 30 1 1	0 0 0 0 0	129.5	129.5	300.01	202.88	4.26	0.51
30 30 30 1 1	30 30 30 1 1	155.875	155.875	300.01	79.80	9.53	0.59
30 30 30 1 1	30 10 5 2 1	142.7917	142.7917	300.13	103.88	**300.00**	0.41
30 30 30 30 1	0 0 0 0 0	207.75	207.75	300.02	47.25	3.09	0.16
30 30 30 30 1	30 30 30 30 1	236.5417	236.5417	300.03	37.03	3.76	0.35
30 30 30 30 1	30 10 5 2 1	218.2917	218.2917	300.16	37.99	74.46	0.21
			Average time	300.064	50.011	57.812	0.382

Table 2.14: Instances with positive maintenance costs ($m = 5, T = 24$)

a	OPT	MIP value	MIP time(s)	SP (s)	new-MIP (s)	FF (s)	NGVNS
1 1 1 1 1 1 1 1 1 1	49	49	300.12	1	**300.00**	**300.00**	0.32
10 9 8 7 6 5 4 3 2 1	232	**232.3333**	300.12	29	**300.00**	**300.00**	0.38
10 10 10 10 10 10 10 10 10 1	413.5	413.5	300.25	2	**300.00**	**300.00**	0.22
100 1 1 1 1 1 1 1 1 1	126.5	126.5	300.08	1	**300.00**	11.71	0.03
1000 1 1 1 1 1 1 1 1 1	576.5	576.5	279.32	7	150.05	0.76	0.05
		Average time	295.798	8.000	270.010	182.494	0.199

Table 2.15: Instances with many machines ($m = 10, T = 18, b_i = 0, i \in M$)

Instances from	MIP (s)	SP (s)	new-MIP (s)	FF (s)	NGVNS (s)
Table 2.11	0.960	46.333	0.176	0.202	0.028
Table 2.12	61.890	2.433	1.207	0.776	0.103
Table 2.13	217.315	812.708	4.940	7.606	2.994
Table 2.15	295.798	8.000	270.010	182.494	0.199
Average time	143.991	217.369	69.083	47.770	0.831

Table 2.16: Average CPU times

method for small instances in Table 2.11 but the best one for the largest instances presented in Table 2.15.

iv) The old MIP model is least reliable. For example in Table 2.14 and Table 2.15, for only one instance (out of 35) the optimal solution has been proven within 300 seconds. However, the optimal solution has not been reached on eight instances (boldfaced values in those tables).

2.4.8 Unit commitment problem

Unit commitment problem (UCP) consists of determining an optimal production plan for a given set of power plants over a given time horizon so the total production cost is minimized, while satisfying various constraints. Every power plant individually needs to satisfy: minimum up time (minimal number of consecutive time periods during which the unit must be turned on), minimum down time (minimal number of consecutive time periods during which unit must be turned off) and production limit constraints (lower and upper production bounds). The total production of all active plants must satisfy the required demand minding that the maximal possible production cannot be less than the sum of required demand and required spinning reserves [225].

Unit commitment problem can be formulated as a mixed integer nonlinear problem. Binary variables represent the ON/OFF state of every unit for each time period, while continuous variables quantify the unit production expressed in megawatts for each time period. It is easy to conclude that the number of all possible solutions grows exponentially by increasing the number of plants. The UCP is NP-hard, which means that it can not be exactly solved in reasonable amount of time. This holds even for moderate number

of units, therefore, many heuristics have been proposed in the literature to solve UCP approximatively.

The exact method based on dynamic programming [117, 120, 157, 177] for solving the UCP was able to tackle only problems with small number of units. Many heuristic and metaheuristic methods have been proposed up to now for the UCP such as: priority list method [27], genetic algorithms [55, 142, 202, 207], tabu search algorithms [185], particle swarm optimization algorithms [196, 229], ant colony algorithms [193], fuzzy logic [66], artificial neural networks [63, 200], evolutionary programming [139], simulated annealing [194, 195, 197]. We propose a hybrid approaches that combine VNS with mathematical programming (i.e., in order to solve economic dispatch subproblem in each time period *Lambda iteration* method is launched) [208, 211].

A solution of UCP is represented by the matrix U, whose each entry U_{ij} represents the state of the unit i in the time period j. Relatively to a solution U we define the set $\mathcal{N}_k^{'}(U)$ of all solutions which can be obtained by changing values exactly k elements of the matrix U. Obviously, such set contains not only feasible solutions, but also solutions that violate some constraints. However, the infeasible solutions may be converted to the feasible ones using procedure proposed in [60]. Hence, let us denote with M the set of all solutions which can be obtained by repairing infeasible solutions from $\mathcal{N}_k^{'}(U)$. Now, we define the k-th neighborhood of solution U (denoted by $\mathcal{N}_k(U)$) as the union of $\mathcal{N}_k^{''}(U)$ and M, where $\mathcal{N}_k^{''}(U)$ represents the set of all feasible solutions from $\mathcal{N}_k^{'}(U)$.

We develop two variants of pipeVND for solving UCP that differ in order of the local searches during the optimization process. However, both pipeVNDs consist of the same set of two neighborhoods sequentially explored one after another. Additionally, each of them are iterated until there is no improvement in the objective function value. The used local searches attempt to decommit units, preserving feasibility. They are based on the priority list, i.e., the search for units which will be decommited is organized according to the descending merit order. The difference between these two local searches is in the number of consecutive time periods (hours) attempted to decommit a unit. The first local search (LS1) attempts to decommit unit i for a period of

T_i^{down} hours (where T_i^{down} equals to minimum down time of the unit i); the second one (LS2) attempts to decommit each unit in one hour. Both local searches explore the solution space using the first improvement strategy. The `pipeVND1` applies LS1 and then LS2, while `pipeVND2` employs LS2 and then LS1.

We developed two variants of GVNS that uses the same shaking procedure for diversification. The shaking procedure returns a random solution from the k-th neighborhood of a given solution U. However, two proposed variants differ in the way of performing intensification. The first one called `GVNS`, uses the `pipeVND1` as a local search while the second one, called `Adaptive_GVNS`, decides whether the `pipeVND1` or the `pipeVND2` will be applied in some iteration of GVNS, depending on their success in previous solution process. In the first step `Adaptive_GVNS` uses `pipeVND1`, while in all other iterations the decision which pipe VND will be applied is made as follows. Initially, both `pipeVND1` and `pipeVND2` have the same merit value $w = 0.5$ assigned to them. After that at each iteration their merits are updated dynamically. Namely, the merit value of the used pipeVND variant is increased or decreased for some value α (e.g., $\alpha = 0.1$) depending on whether the currently best found solution is improved or not in that iteration. The currently used pipeVND will be replaced by another in the next iteration if its merit becomes negative. If replacement of a pipeVND occurs, its merit is reset to the initial value w.

Both `GVNS` and `Adaptive_GVNS`, are tested by constructing initial solutions in two different ways. If the initial solution is obtained by greedy procedure [60], the corresponding `GVNS` and `Adaptive_GVNS` variants are denoted by `GVNS-G` and `Adaptive_GVNS-G`, respectively. Similarly, if `GVNS` (`Adaptive_GVNS`) uses greedy randomized initial solution [72], that variants are named as `GVNS-R` (`Adaptive_GVNS-R`). The greedy randomized initial solutions for both GVNSs are built iteratively, starting from the solution created by the greedy procedure. Each iteration consists of choosing a random solution from the first neighborhood of the current solution and setting the chosen solution to be a new current solution. The whole process is repeated $N \cdot T$ times (where N and T are the number of units and the number of time periods, respectively).

Computational results

To perform empirical analysis we use the following two data sets from the literature:
Case study 1 [142]. This case study contains test instances with up to 100 units. The instances with more than 10 units are derived duplicating the data of the basic instance with 10 units. The load demands for those derived instances are adjusted in proportion to the number of units. The spinning reserve requirement, for all instances, is set to 10% of total load demand.
Case study 2 [132]. The second case study consists of 38 generating units from the practical Taiwan Power (Taipower). The spinning reserve requirement is set to 11% of the total load demand.

Comparison with other heuristics on Case study 1: The fuel costs obtained by our methods are compared with fuel costs obtained by the following 23 heuristics from the literature: Lagrangian relaxation (LR) [142]; genetic algorithm (GA) [142]; enhanced adaptive Lagrangian relaxation (ELR) [142]; Dynamic Programming with ELR (DPLR) [142]; Lagrangian relaxation and genetic algorithm (LRGA) [38]; genetic algorithm based on characteristic classification (GACC) [202]; evolutionary programming (EP) [139]; priority-list-based evolutionary algorithm (PLEA) [205]; extended priority list (EPL) [205]; integer coded genetic algorithm (ICGA) [55]; a Lagrangian multiplier based sensitive index to determine the unit commitment of thermal units (LMBSI) [203]; improved pre-prepared power demand and Muller method (IPPDTM) [37]; quantum inspired binary particle swarm optimization (QBPSO) [137]; quantum-inspired evolutionary algorithms (QEA-UC) [150] and (IQEA-UC) [41]; shuffled frog leaping algorithm (SFLA) [64]; imperialistic competition algorithm (ICA) [171]; gravitational search algorithm (GSA) [192]; semi-definite programming (SDP) [135, 162]; tighter relaxation method (RM) [184]. Comparative results are given in Table 2.17. It should be emphasized that for test instance with 20 units, LRGA, SFLA and GSA heuristics report solution values better than the optimal solution value (which equals to 1123297 see [219]). Therefore, we boldfaced that value in Table 2.17, but values better than optimal present in italic font. For instance with

40 units the optimal solution is not known. Hence, it is questionable if the value of 2242178 obtained by LRGA is really reliable (since it reported better value than optimal for $N = 20$).

No. of Units	10 TU's	20 TU's	40 TU's	60 TU's	80 TU's	100 TU's	
Method	Operating Cost($)						Average
LR [142]	565825	1130660	2258503	3394066	4526022	5657277	2922058.83
ELR [175]	563977	**1123297**	2244237	3363491	4485633	5605678	2897718.83
LRGA [38]	564800	*1122622*	**2242178**	3371079	4501844	5613127	2902608.33
DPLR [175]	564049	1128098	2256195	3384293	4512391	5640488	2914252.33
GA [142]	565825	1126243	2251911	3376625	4504933	5627437	2908829.00
GACC [202]	563977	1125516	2249715	3375065	4505614	5626514	2907733.50
EP [139]	564551	1125494	2249093	3371611	4498479	5623885	2905518.83
ICGA [55]	566404	1127244	2254123	3378108	4498943	5630838	2909276.67
PLEA [205]	563977	1124295	2243913	3363892	4487354	5607904	2898555.83
EPL [205]	563977	1124369	2246508	3366210	4489322	5608440	2899804.33
LMBSI [203]	563977	1123990	2243708	3362918	4483593	5602844	2896838.33
IPPDTM [37]	563977	-	2247162	3366874	4490208	5609782	-
QBPSO [137]	563977	**1123297**	2242957	3361980	4482085	5602486	2896130.33
QEA-UC [150]	**563938**	1123607	2245557	3366676	4488470	5609550	2899633.00
IQEA-UC[41]	**563938**	**1123297**	2242980	3362010	4482826	5602387	2896239.67
SFLA[64]	564769	*1123261*	2246005	3368257	4503928	5624526	2905124.33
ICA [171]	**563938**	1124274	2247078	3371722	4497919	5617913	2903807.33
GSA[192]	**563938**	*1123216*	2242741	3362447	4483864	5600883	2896181.50
SDP[135]	563938	1124357	2243328	3363031	4484365	5602538	2896926.17
SDP[162]	563977	1124410	2243144	3360512	4480652	5598727	2895237.00
RM [184]	563977	1123990	2243676	3361589	4481833	5599761	2895804.33
`GVNS-R`	563938	**1123297**	2242882	3360316	4480515	**5597962**	2894818.33
`GVNS-G`	563938	**1123297**	2242882	3360699	4480617	5600133	2895261.00
`Adaptive_GVNS-R`	563938	**1123297**	**2242596**	**3360181**	**4480328**	5597964	2894717.33
`Adaptive_GVNS-G`	563938	**1123297**	2242882	3361119	4480617	5598876	2895121.50

Table 2.17: Comparison on Case Study 1 [142]

From results presented in Table 2.17 the following conclusion may be drawn:

- All proposed GVNS variants succeed in finding optimal solutions for instances with up to 20 units
- For instances with 60 and 80 units, `Adaptive_GVNS-R` provides solutions of better quality than all proposed heuristics up to now in the literature. On the other hand, `GVNS-R` offers the best solution for instance with 100 units .

- Regarding the average solution cost achieved by each of compared heuristics, we conclude that `Adaptive_GVNS-R` outperforms all the others. The second best heuristic, turns to be `GVNS-R`, while `Adaptive_GVNS-G` takes the third place in the overall ranking. `GVNS-G` is ranked as the fifth best immediately behind SDP heuristic [162].

The execution times of `GVNS-R`, `GVNS-G`, `Adaptive_GVNS-R` and `Adaptive_GVNS-G` as well as execution times of all other methods, are presented in Table 2.18. Note that the computer configurations for the methods of LMBSI [203], IPPDTM [37], QBPSO [137], QEA-UC [150], IQEA-UC [41], GSA [192], ICA [171], SDP [135], RM [184] are 2 GHz CPU, Pentium IV 2.8 GHz, Pentium IV 2.0 GHz, Intel Core 2.39 GHz, Intel core 2 Duo CPU 2.66 GHz, Intel Pentium IV 2-GHz CPU, Intel Core 2 Quad 2.4 GHz, core 2 duo processor 2 GHz, Intel Core 2 Duo Processor T5300 1.73 GHz and AMD Dual-Core 4800 + 2.5 GHz, respectively. All proposed GVNSs have been run on a computer with Intel i7 2.8 GHz CPU. All in all computer platforms have the similar characteristics. Note that our code is run on a single processor while some other use more parallel CPUs.

No. of Units	10 TU's	20 TU's	40 TU's	60 TU's	80 TU's	100 TU's	
Method	CPU time(s)						Average
LMBSI [203]	10.00	18.00	27.00	40.00	54.00	73.00	37.00
IPPDTM [37]	0.52	-	6.49	17.39	31.23	46.55	20.44
QBPSO [137]	18.00	50.00	158.00	328.00	554.00	833.00	323.50
QEA-UC [150]	19.00	28.00	43.00	54.00	66.00	80.00	48.33
IQEA-UC [41]	34.00	98.00	146.00	191.00	235.00	293.00	166.17
ICA [171]	48.00	63.00	151.00	366.00	994.00	1376.00	499.67
GSA [192]	2.89	13.72	74.66	103.41	146.45	204.93	91.01
SDP [135]	25.41	63.94	157.73	260.76	353.84	392.56	209.04
RM [184]	1.15	2.14	4.83	8.79	13.02	17.10	7.84
`GVNS-R`	0.23	2.46	63.19	126.82	22.56	374.49	98.29
`GVNS-G`	0.05	2.5	6.64	436.35	98.76	351.56	149.31
`Adaptive_GVNS-R`	0.08	1.95	109.85	212.75	64.86	283.84	112.22
`Adaptive_GVNS-G`	0.04	1.23	2.14	109.53	287.49	552.49	158.82

Table 2.18: CPU time: Case Study 1 [142]

Comparison with other heuristics on Case study 2: In order to perform the comparison of our methods with other heuristic approaches on this

data set, the start up cost in the first hour is neglected as in [37]. The results obtained by the heuristics from the literature are compared with our methods in Table 2.19: dynamic programming (DP) [132], Lagrangian relaxation (LR) [132], simulated annealing (SA) [132], constrained logic programming (CLP) [132], fuzzy optimization (FO) [66], matrix real coded genetic algorithm (MRCGA) [206], memory bounded ant colony optimization (MACO) [193], fuzzy adaptive particle swarm optimization(FAPSO) [196], absolutely stochastic simulated annealing (ASSA) [194], twofold simulated annealing (TFSA) [197], heuristic and ASA (HASSA) [195], enhanced merit order and augmented Lagrange Hopfield network (EMO-ALHN) [61], improved preprepared power demand and Muller method (IPPDTM) [37] and Augmented Lagrange hopfield network based Lagrangian relaxation (ALHN-LR) [62].

DP [132], LR [132], SA [132], and CLP [132] were executed on 486-66 PC, MRCGA [206] on Intel Celeron 1.2 GHz, ASSA [194] on Intel Pentium 4 1.4 GHz CPU, TFSA [197] and HASSA [195] on Intel(R) Celeron(TM) CPU, EMO-ALHN [61] on Intel Cleron 1.1 GHz, IPPDTM [37] on Pentium IV 2.8 GHz, ALHN-LR [62] on Intel Celeron 1.5 GHz. There is no report of computer used for the FO, MACO and FAPSO methods. GVNS approaches have been run on a computer with Intel i7 2.8 GHz CPU.

Among heuristics mentioned above, some have been tested on the same 38-units system, but with increased operating time. Namely, the total time of 24 hours has been extended to 72 and 168 hours. The increased load demands are adapted naturally. Such cases are compared in columns 3 and 4 of Table 2.19, i.e., only the costs of HASSA, EMO-ALHN, ALHN-LR and GVNS methods are given. In order to perform a fair comparison with previous approaches, maximum CPU time allowed to be consumed by our GVNS methods were set to 10 seconds for time horizon of 24 hours, 20 seconds for time horizon of 72 hours and 40 seconds for time horizon of 168 hours. However, in Table 2.20 we present results obtained by our GVNS methods extending the time limits to 600 seconds for each time horizon.

Computational results show that our methods, for any time horizon, provide better quality solutions than those obtained by previously proposed methods. It should be noted that for any time horizon solutions offered by `Adaptive_GVNS-G` are better than those found by other GVNS based methods.

This result could be explained by the fact that the solution space is enormous, therefore it is important to start the exploration from a reasonably good initial solution (i.e., greedy solution) in order to get high quality solution within the imposed time limit. Additionally, the adaptive mechanism embedded within GVNS turns out to be powerful enough to help GVNS to provide high-quality solutions in a reasonable amount of time.

Time horizon	24h	72h	168h	24h	72h	168h
Method	Operating Cost(M$)			CPU time(s)		
DP [132]	210.5	-	-	24.00	-	-
LR [132]	209	-	-	7.00	-	-
SA [132]	207.8	-	-	1690.00	-	-
CLP [132]	208.1	-	-	10.00	-	-
FO [66]	207.8	-	-	-	-	-
MRCGA [206]	204.6	-	-	-	-	-
MACO [193]	200.46	-	-	111.90	-	-
FAPSO [196]	196.73	-	-	6.07	-	-
ASSA [194]	196.7	-	-	3.96	-	-
TFSA [197]	197.98	-	-	3.43	-	-
IPPDTM [37]	196.06	-	-	1.36	-	-
HASSA [195]	196.96	601.4	1410.47	5.01	9.04	37.64
EMO-ALHN [61]	197.5	590.66	1376.55	0.21	0.66	1.91
ALHN-LR [62]	195.87	585.27	1366.18	8.64	13.58	16.21
`GVNS-R`	194.44	583.62	1365.48	7.18	19.55	39.54
`GVNS-G`	194.05	583.71	1363.74	9.94	9.62	39.10
`Adaptive_GVNS-R`	194.16	583.76	1364.92	9.92	11.98	39.85
`Adaptive_GVNS-G`	**193.94**	**583.32**	**1362.41**	9.67	19.52	39.59

Table 2.19: Computational Results: Case Study 2 [132]

Time horizon	24h	72h	168h	24h	72h	168h
Method	Operating Cost(M$)			CPU time(s)		
`GVNS-R`	**193.75**	581.58	1360.35	416.36	597.36	598.35
`GVNS-G`	**193.75**	582.26	1360.45	538.17	491.58	589.48
`Adaptive_GVNS-R`	**193.75**	**581.57**	1358.66	541.39	450.51	592.62
`Adaptive_GVNS-G`	**193.75**	**581.57**	**1358.65**	359.63	390.17	539.59

Table 2.20: Computational Results with time limit set to 600s: Case Study 2 [132];

2.5 Concluding remarks

This chapter has been focused on pure Variable Neighborhood Search (VNS) based heuristics, i.e, those that are not obtained combining VNS with other metaherutistics or mathematical programming approaches. However, the hybrid approaches that combine VNS and mathematical programming technics will be presented in Chapter 3. More precisely, in this chapter we described main ingredients of a VNS heuristic as well as several its variants. Some of these variants had been proposed before, while the other were proposed by the author for the first time. The newly proposed VNS variant such as a two-level VNS, a nested VNS, and a cyclic variable neighborhood descent were successfully applied for solving several NP-hard problems. Besides that, in this chapter we provide a review of successful applications of VNS heuristics, developed by the author, for solving variety optimization problems arising in transportation, logistic, scheduling, power generation and clustering such as: Traveling salesman problem with time windows [169], Attractive traveling salesman problem [170], Traveling salesman problem with draft limits [210], Swap-body vehicle routing problem [212], Uncapacitated r-allocation p-hub median problem [213], Minimum sum-of-squares clustering on networks [36], Periodic maintenance problem [209] and Unit commitment problem [211]. The computational results obtained solving the considered problems, demonstrate the superiority of the proposed VNS heuristics over the previously proposed heuristics. In other words, the proposed VNS heuristics turn out to be the new state-of-the-art heuristics. Such performances of VNS heuristics indicate that developing VNS heuristics for solving other NP-hard optimization problems is promising research avenue.

Chapter 3

Matheuristics for 0–1 Mixed Integer Program

3.1 Introduction

The approaches used to tackle the optimization problems may be divided on exact and heuristic. The exact methods (e.g., branch-and-bound, branch-and-cut, dynamic programming etc.) are topic of mathematical programming area and such methods can generate provably optimal solutions to optimization problems. On the other hand, heuristic approaches are able to generate "good" solutions to optimization problems but not necessarily provably optimal. Since many optimization problems cannot be solved to optimality within acceptable time/resource consumption, while heuristics can not provide optimality of a generated solution, researches started to develop so-called matheuristics in order to possibly overcome these issues. Matheuristics are heuristic algorithms made by the combining metaheuristics and mathematic programming techniques. An essential feature of a matheuristic is the exploitation of features derived from the mathematical model of the considered problem in certain stages of the solution process. For surveys on matheuristics we refer to Ball (2011) [13] and Maniezzo et al. (2010) [160].

Due to the recent advances in exact methods for mixed integer linear programming (MIP) problems many researchers designed matheuristics that incorporate phases where MIP are solved solved exactly. Following this research line we propose two new diving heuristics for finding a feasible solution for a MIP problem, called *Variable Neighborhood* (VN) diving and *Single Neighborhood* (SN) diving, respectively. They perform systematic hard variable fixing (i.e., diving) by exploiting the information obtained from a series of LP relaxations in order to generate a sequence of subproblems.

Pseudo cuts are added during the search process to avoid revisiting the same search space areas. VN diving is based on the variable neighborhood decomposition search framework [122]. Conversely, SN diving explores only a single neighborhood in each iteration: if a feasible solution is not found, then the next reference solution is chosen using the feasibility pump principle [77] and the search history [92]. Moreover, we prove that the two proposed algorithms converge in a finite number of iterations. Additionally, we propose several iterative linear programming-based heuristics for solving Fixed-Charge Multicommodity Network Design (MCND) problem [159]. We propose how to adapt well-known Slope Scaling heuristic [51] for MCND in order to tackle reduced problems of MCND obtained by fixing some binary variables. Moreover, we show that ideas of a convergent algorithm based on the LP-relaxation and pseudo-cuts may be used to guide a Slope Scaling heuristic during the search for an optimal (or near-optimal) solution and vice-versa.

The rest of the chapter is organized as follows. Section 3.2 is devoted to describing Variable Neighborhood diving and Single Neighborhood diving heuristics, while in Section 3.2 we describe iterative linear programming-based heuristics for solving Fixed-Charge Multicommodity Network Design problem. More precisely, Section 3.2 starts by giving brief introduction to MIP problems and methods for generating first feasible solution. After that, in Subsection 3.2.1 the notation used throughout the section is presented. A detailed description of the two new diving heuristics for MIP feasibility is provided in Subsection 3.2.2. In Subsection 3.2.3, we analyze the performance of the proposed methods as compared to the commercial IBM ILOG CPLEX 12.4 MIP solver and the basic and objective variant of the FP heuristic [2, 77]. On the other hand, Section 3.3 starts by describing Fixed-Charge Multicommodity Network Design problem, followed by the description of Slope Scaling heuristics in Subsection 3.3.1. In Subsection 3.3.2 Convergent algorithm based on linear programming relaxation and pseudo cuts is presented. Subsection 3.3.3 contains an adaptation of Slope scaling heuristic for solving reduced problems of MCND and the description of several iterative linear programming-based heuristics for solving MCND. In Subsection 3.3.3 we compare proposed approaches with previously proposed ones in order to asses their quality. Finally, in Section 3.4 we draw some conclusions. For the

sake of readability of the chapter, the notation used will be presented always when needed although that may cause overlapping with the notation used in the previous chapters.

3.2 Variable and Single Neighborhood Diving for MIP Feasibility

The mixed integer programming (MIP) problem can be formulated as follows:

$$(P) \qquad \min\{cx \mid x \in X\}, \tag{3.1}$$

where $X = \{x \in \mathbb{R}^n \mid Ax \leq b, x_j \in \{0,1\} \text{ for } j \in \mathcal{B}, x_j \in \mathbb{Z}^+ \text{ for } j \in \mathcal{G}, l_j \leq x_j \leq u_j \text{ for } j \in \mathcal{C} \cup \mathcal{G}\}$ ($\mathcal{B}, \mathcal{G}, \mathcal{C}$ respectively constitute the index sets for the binary (0-1), integer (non-binary) and continuous variables) is the feasible set, cx is the objective function, and $x \in X$ are the feasible solutions. In the special case when $\mathcal{G} = \emptyset$, the resulting MIP problem is called the 0-1 MIP problem (0-1 MIP). The LP-relaxation of problem P, denoted as LP(P), is obtained from the original formulation by relaxing the integer requirements on x:

$$\text{LP}(P) \qquad \min\{cx \mid x \in \overline{X}\}, \tag{3.2}$$

where $\overline{X} = \{x \in \mathbb{R}^n \mid Ax \leq b, l_j \leq x_j \leq u_j \text{ for } j \in \mathcal{G} \cup \mathcal{C}, x_j \in [0,1] \text{ for } j \in \mathcal{B}\}$.

Many real-world problems can be modeled as MIP problems [25, 43]. However, a number of special cases of MIP problem are proven to be NP-hard [80] and cannot be solved to optimality within acceptable time/space with existing exact methods. This is why various heuristic methods have been designed in attempt to find good near-optimal solutions of hard MIP problems. Most of them start from a given feasible solution and try to improve it. Still, finding a feasible solution of 0-1 MIP is proven to be NP-complete [224] and for a number of instances finding a feasible solution remains hard in practice. This calls for the development of efficient constructive heuristics which can attain feasible solutions in short time. Over the last decade, a number of heuristics that address the problem of MIP feasibility have been proposed such as the feasibility Pump (FP) for the special case of pure 0-1 MIP problem

in [77]; FP for the case of general MIP problems [18]; the objective FP [2], the local branching [75] and so on. For detailed description of these heuristics see Chapter 1.

The concept of variable fixing in order to find solutions to MIP problems was conceived in the late 1970s and early 1980s, when the first methods of this type were proposed [9, 204]. Subproblems are iteratively generated by fixing a certain number of variables in the original problem according to the solution of the linear programming relaxation of the original problem. This approach is also referred to as a *core approach*, since the subproblems so obtained are sometimes called *core problems* [9, 183]. The used terms *hard variable fixing* or *diving* are also present in the literature (see, for example, [56]). The critical issue in this type of methods is the way in which the variables to be fixed are chosen. Depending on the selection strategy and the way of manipulating the obtained subproblems, different MIP solution methods are obtained. The basic strategy was initially proposed in [9], for solving the multidimensional knapsack problem. A number of its successful extensions were proposed over the years. For example, a greedy strategy for determining the core is developed in [181], whereas in [183] the core is defined according to a chosen efficiency function. Another iterative scheme, again for the 0-1 multidimensional knapsack problem, was developed in [223]. This scheme, which is based on a dynamic fixation of the variables, uses the search history to build up feasible solutions and to select variables for a permanent/temporary fixation. Variable neighborhood search was combined with a very large scale neighborhood search approach to select variables for fixing (*binding sets*) for the general assignment problem [164, 166]. This approach was further extended for 0-1 mixed integer programming in general [165].

With the expansion of general-purpose MIP solvers over the last decade, different hybridizations of MIP heuristics with commercial solvers are becoming increasingly popular. A number of efficient heuristics that perform some kind of variable fixing at each node of the Branch and Bound tree in the CPLEX MIP solver have been developed such as, e.g., Relaxation induced neighborhood search (RINS) [56], Relaxation enforced neighborhood search [20] and variable neighborhood decomposition search [122] heuristic proposed

in [151]. For extensive description of aforementioned heuristics see Chapter 1 of this thesis.

We propose two new diving heuristics for MIP feasibility, which exploit the information obtained from a series of LP relaxations. Since the variables to be fixed depend on the LP relaxation solution values, this approach may also be called *relaxation guided diving*. Relaxation guided variable neighborhood search was proposed in [182], but for defining the order of neighborhoods within VNS (where neighborhoods are defined by soft variable fixing) rather than selecting the variables to be hard-fixed.

The first heuristic, called *variable neighborhood diving* is based on the variable neighborhood decomposition search principle [122]. A similar approach was proposed in [151] for optimizing 0-1 MIP problems starting from a given initial MIP feasible solution. We propose a modification of the algorithm from [151] for constructing feasible solutions of 0-1 MIP problems. We exploit the fact that the CPLEX MIP solver can be used not only for finding near-optimal solutions but also as a black-box for finding a first feasible solution for a given 0-1 MIP problem. We also extend this approach for general MIP problems, so that fixation is performed on general integer variables as well. The second heuristic, called *single neighborhood diving* explores only a single neighborhood in each iteration. However, the size of the neighborhood is updated dynamically according to the solution status of the subproblem in a previous iteration. The incumbent solution is updated in a feasibility pump manner, whereas revisiting the same point in the search process is prohibited by keeping the list of all visited reference solutions. This list is implemented as a set of constraints in a new (dummy) MIP problem. We show that our proposed algorithms significantly outperform the CPLEX 12.4 MIP solver and the recent variants of the feasibility pump heuristic, both regarding the solution quality and the computational time.

3.2.1 Notation

Given an arbitrary integer solution x' of problem (3.1) and an arbitrary subset $J \subseteq \mathcal{B} \cup \mathcal{G}$ of integer variables, the problem *reduced* from the original problem

P and associated with x' and J can be defined as:

$$P(x', J) \qquad \min\{cx \mid x \in X, x_j = x'_j \text{ for } j \in J\} \tag{3.3}$$

If C is a set of constraints, we will denote with $(P \mid C)$ the problem obtained by adding all constraints in C to the problem P. Let x and y be two arbitrary integer solutions of the problem P. The distance between x and y is then defined as

$$\Delta(x, y) = \sum_{j \in \mathcal{B} \cup \mathcal{G}} |x_j - y_j| \tag{3.4}$$

If $J \subseteq \mathcal{B} \cup \mathcal{G}$, the partial distance between x and y, relative to J, is defined as $\Delta(J, x, y) = \sum_{j \in J} \mid x_j - y_j \mid$ (obviously, $\Delta(\mathcal{B} \cup \mathcal{G}, x, y) = \Delta(x, y)$). The linearization of the distance function $\Delta(x, y)$, as defined in (3.4), requires the introduction of additional variables. More precisely, for any integer feasible vector y, function $\Delta(x, y)$ can be linearized as follows [74]:

$$\Delta(x, y) = \sum_{j \in \mathcal{B} \cup \mathcal{G}: y_j = l_j} (x_j - l_j) + \sum_{j \in \mathcal{B} \cup \mathcal{G}: y_j = u_j} (u_j - x_j) + \sum_{j \in \mathcal{G}: l_j < y_j < u_j} d_j, \tag{3.5}$$

where $l_j = 0$ and $u_j = 1$ for $j \in \mathcal{B}$ and new variables $d_j = |x_j - y_j|$ need to satisfy the following constraints :

$$d_j \geq x_j - y_j \text{ and } d_j \geq y_j - x_j \text{ for all } j \in \{i \in \mathcal{G} \mid l_i < y_i < u_i\}. \tag{3.6}$$

In the special case of 0-1 MIP problems, the distance function between any two binary vectors x and y can be expressed as:

$$\delta(x, y) = \sum_{j \in \mathcal{B}} x_j(1 - y_j) + y_j(1 - x_j). \tag{3.7}$$

Furthermore, if x is a given binary vector, then formula (3.7) can be used to compute the distance from x to any vector $\overline{x} \in \mathbb{R}^n$:

$$\delta(x, \overline{x}) = \sum_{j \in \mathcal{B}} x_j(1 - \overline{x}_j) + \overline{x}_j(1 - x_j).$$

As in the case of general MIP problems, the partial distance between x and

$\overline{x}$, relative to $J \subseteq \mathcal{B}$, is defined as $\delta(J, x, \overline{x}) = \sum_{j \in J} x_j(1 - \overline{x}_j) + \overline{x}_j(1 - x_j)$. Note that the distance function δ, as defined in (3.7), can also be used for general MIP problems, by taking into account that $\delta(x, y) = \Delta(\mathcal{B}, x, y)$ for any two solution vectors x and y of a general MIP problem (3.1).

The LP-relaxation of the modified problem, obtained from a MIP problem P, as defined in (3.1), by replacing the original objective function cx with $\delta(\tilde{x}, x)$, for a given integer vector $\tilde{x} \in \{0,1\}^{|\mathcal{B}|} \times \mathbb{Z}_+^{|\mathcal{G}|} \times \mathbb{R}_+^{|\mathcal{C}|}$, can be expressed as:

$$\mathrm{LP}(P, \tilde{x}) \qquad \min\{\delta(\tilde{x}, x) \mid x \in \overline{X}\} \tag{3.8}$$

Similarly, the notation $\mathrm{MIP}(P, \widetilde{x})$ will be used to denote a modified problem, obtained from P by replacing the original objective function with $\delta(x, \widetilde{x})$:

$$\mathrm{MIP}(P, \tilde{x}) \qquad \min\{\delta(\tilde{x}, x) \mid x \in X\}. \tag{3.9}$$

We will also define the *rounding* $[x]$ of any vector x, as vector $[x] = ([x]_j)$, with:

$$[x]_j = \begin{cases} \lfloor x_j + 0.5 \rfloor, & j \in \mathcal{B} \cup \mathcal{G} \\ x_j, & j \in \mathcal{C}. \end{cases} \tag{3.10}$$

The neighborhood structures $\{\mathcal{N}_k \mid 1 \leq k_{min} \leq k \leq k_{max} \leq |\mathcal{B}| + |\mathcal{G}|\}$ 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 $x \in X$ is defined as

$$\mathcal{N}_k(x) = \{y \in X \mid \delta(x, y) \ = \ k\}. \tag{3.11}$$

3.2.2 New Diving Heuristics for MIP Feasibility

The new diving heuristics presented in this section are based on the systematic hard variable fixing (diving) process, according to the information obtained from the LP relaxation solution of the problem. They rely on the observation that a general-purpose MIP solver can be used not only for finding (near) optimal solutions of a given input problem, but also for finding the initial feasible solution. For the sake of simplicity, we will first present both algorithms for the special case of 0-1 MIP problems. After that we will explain how the presented algorithms can be adapted for solving general MIP problems.

Variable Neighborhood Diving

The variable neighborhood (VN) diving algorithm begins by obtaining the LP-relaxation solution $\overline{x}$ of the original problem P and generating an initial integer (not necessarily feasible) solution $\tilde{x} = [\overline{x}]$ by rounding the LP-solution $\overline{x}$. If the optimal solution $\overline{x}$ is integer feasible for P, we stop and return $\overline{x}$. At each iteration of the VN diving procedure, we compute the distances $\delta_j = \mid \tilde{x}_j - \overline{x}_j \mid$ from the current integer solution values $(\tilde{x}_j)_{j \in \mathcal{B}}$ to the corresponding LP-relaxation solution values $(\overline{x}_j)_{j \in \mathcal{B}}$ and index the variables $\tilde{x}_j, j \in \mathcal{B}$ so that $\delta_1 \leq \delta_2 \leq \ldots \leq \delta_{|\mathcal{B}|}$. Then, we successively solve the subproblems $P(\tilde{x}, \{1, \ldots, k\})$ obtained from the original problem P, where the first k variables are fixed to their values in the current incumbent solution $\tilde{x}$. If a feasible solution is found by solving $P(\tilde{x}, \{1, \ldots, k\})$, it is returned as a feasible solution of the original problem P. Otherwise, a pseudo-cut $\delta(\{1, \ldots, k\}, \tilde{x}, x) \geq 1$ is added in order to avoid exploring the search space of $P(\tilde{x}, \{1, \ldots, k\})$ again, and the next subproblem is examined. If no feasible solution is detected after solving all subproblems $P(\tilde{x}, \{1, \ldots, k\})$, $k_{min} \leq k \leq k_{max}$, $k_{min} = k_{step}$, $k_{max} = |\mathcal{B}| - k_{step}$, the linear relaxation of the current problem P, which includes all the pseudo-cuts added during the search process, is solved and the process is iterated. If no feasible solution has been found due to the fulfillment of the stopping criteria, the algorithm reports failure and returns the last (infeasible) integer solution.

The pseudo-code of the proposed VN diving heuristic is given in Algorithm 37. The input parameters for the VN diving algorithm are the input MIP problem P and the parameter d, which controls the change of neighborhood size during the search process. In all pseudo-codes, a statement of the form $y =$ `FindFirstFeasible`$(P, time)$ denotes a call to a generic MIP solver, an attempt to find a first feasible solution of an input problem P within a given time limit *time*. If a feasible solution is found, it is assigned to the variable y, otherwise y retains its previous value.

Since the VN diving procedure examines only a finite number of subproblems, it is easy to prove the following proposition.

Proposition 3.2.1 *If the* `timeLimit` *parameter is set to infinity, the variable neighborhood diving algorithm finishes in a finite number of iterations and*

Algorithm 37: Variable neighborhood diving for 0-1 MIP feasibility.

Function `VNdiving`(P, d);
Set *proceed*1 = `true`, *proceed*2 = `true`; Set *timeLimit* for subproblems;
while *proceed*1 **do**
  Solve the LP relaxation of P to obtain an optimal LP basic solution $\overline{x}$;
  $\tilde{x} = [\overline{x}]$;
  if $\overline{x} = \tilde{x}$ **then return** $\tilde{x}$;
  $\delta_j =\mid \tilde{x}_j - \overline{x}_j \mid$; index x_j so that $\delta_j \leq \delta_{j+1}$, $j = 1, \ldots, |\mathcal{B}| - 1$;
  Set $n_d =\mid \{j \in \mathcal{B} \mid \delta_j \neq 0\} \mid$, $k_{step} = [n_d/d]$, $k = |\mathcal{B}| - k_{step}$;
  while *proceed*2 **and** $k \geq 0$ **do**
    $J_k = \{1, \ldots, k\}$; $x' =$ `FindFirstFeasible`$(P(\tilde{x}, J_k), timeLimit)$;
    if $P(\tilde{x}, J_k)$ *is proven infeasible* **then** $P = (P \mid \delta(J_k, \tilde{x}, x) \geq 1)$;
    if x' is feasible **then return** x';
    if $k - k_{step} > |\mathcal{B}| - n_d$ **then** $k_{step} = \max\{[k/2], 1\}$;
    Set $k = k - k_{step}$;
    Update *proceed*2;
  end
  Update *proceed*1;
end
Output message: "No feasible solution found"; **return** $\tilde{x}$;

either returns a feasible solution of the input problem, or proves the infeasibility of the input problem.

Note however that, in the worst case, the last subproblem examined by VN diving is the original input problem. Therefore, the result of Proposition 3.2.1 does not have any theoretical significance.

Single Neighborhood Diving

In the case of variable neighborhood diving, a set of subproblems $P(\tilde{x}, J_k)$, for different values of k, is examined in each iteration until a feasible solution is found. In the single neighborhood diving procedure, we only examine one subproblem $P(\tilde{x}, J_k)$ in each iteration (a single neighborhood, see Algorithm 38). However, because only a single neighborhood is examined, additional diversification mechanisms are required. This diversification is

provided through keeping the list of constraints which ensures that the same reference integer solution $\widetilde{x}$ cannot occur more than once (i.e., in more than one iteration) in the solution process. An additional MIP problem Q is introduced to store these constraints. In the beginning of the algorithm, Q is initialized as an empty problem (see line 4 in Algorithm 38). Then, in each iteration, if the current reference solution $\widetilde{x}$ is not feasible (see line 8 in Algorithm 38), constraint $\delta(\widetilde{x}, x) \geq \lceil \delta(\widetilde{x}, \overline{x}) \rceil$ is added to Q (line 9). This guarantees that future reference solutions can not be the same as the current one, since the next reference solution is obtained by solving the problem MIP$(Q, [\overline{x}])$ (see line 17), which contains all constraints from Q, (see definition (3.9)). The variables to be fixed in the current subproblem are chosen among those which have the same value as in the linear relaxation solution of the modified problem LP$(P, \widetilde{x})$, where $\widetilde{x}$ is the current reference integer solution (see lines 7 and 11). The number of variables to be fixed is controlled by the parameter α (line 11). After initialization (line 5), the value of α is updated in each iteration, depending on the solution status returned from the MIP solver. If the current subproblem is proven infeasible, the value of α is increased in order to reduce the number of fixed variables in the next iteration (see line 16), and thus provide better diversification. Otherwise, if the time limit allowed for subproblem is exceeded without reaching a feasible solution or proving the subproblem infeasibility, the value of α is decreased. Decreasing the value of α, increases the number of fixed variables in the next iteration (see line 17), and thus reduces the size of the next subproblem. In the feasibility pump, the next reference integer solution is obtained by simply rounding the linear relaxation solution $\overline{x}$ of the modified problem LP$(P, \widetilde{x})$. However, if $[\overline{x}]$ is equal to some of the previous reference solutions, the solution process is caught in a cycle. In order to avoid this type of cycling, we determine the next reference solution as the one which is at the minimum distance from $[\overline{x}]$ (with respect to binary variables) and satisfies all constraints from the current subproblem Q (see line 18). This way we guarantee the convergence of the variable neighborhood diving algorithm, as stated in the

following proposition.

Algorithm 38: Single neighborhood diving for 0-1 MIP feasibility.

Function `SNDiving`(P);
1 Solve the LP relaxation of P to obtain an optimal LP basic solution $\overline{x}$;
2 Set $i = 0$; Set $\widetilde{x}^0 = [\overline{x}]$;
3 **if** $(\overline{x} = \widetilde{x}^0)$ **then return** $\widetilde{x}^0$;
4 Set $Q_0 = \emptyset$;
5 Set *proceed* = `true`; Set *timeLimit* for subproblems; Set value of α;
6 **while** *proceed* **do**
7 Solve the $LP(P, \widetilde{x}^i)$ problem to obtain an optimal solution $\overline{x}$;
8 **if** $(\lceil \delta(\widetilde{x}^i, \overline{x}) \rceil = 0)$ **then return** $\widetilde{x}^i$;
9 $Q_{i+1} = (Q_i \mid \delta(\widetilde{x}^i, x) \geq \lceil \delta(\widetilde{x}^i, \overline{x}) \rceil)$;
10 $\delta_j = \mid \tilde{x}_j - \overline{x}_j \mid$; index x_j so that $\delta_j \leq \delta_{j+1}$, $j = 1, \ldots, |\mathcal{B}| - 1$;
11 $k = [\mid \{j \in \mathcal{B} : \widetilde{x}^i_j = \overline{x}_j\} \mid /\alpha]$; $J_k = \{1, \ldots, k\}$;
12 $x' =$ `FindFirstFeasible`$(P(\widetilde{x}^i, J_k), \mathit{timeLimit})$;
13 **if** *feasible solution found* **then return** x';
14 **if** $P(\widetilde{x}^i, J_k)$ is proven infeasible **then**
15 $Q_{i+1} = (Q_{i+1} \mid \delta(J_k, \widetilde{x}^i, x) \geq 1)$; $P = (P \mid \delta(J_k, \widetilde{x}^i, x) \geq 1)$;
16 $\alpha = 3\alpha/2$;
else
17 **if** *time limit for subproblem exceeded* **then** $\alpha = \max\{1, \alpha/2\}$;
end
18 $\widetilde{x}^{i+1} =$ `FindFirstFeasible`$(\mathrm{MIP}(Q_{i+1}, [\overline{x}]), \mathit{timeLimit})$;
19 **if** $\mathrm{MIP}(Q_{i+1}, [\overline{x}])$ *is proven infeasible* **then** Output message: "Problem P is proven infeasible"; **return**;
20 $i = i + 1$;
end

Proposition 3.2.2 *If the* `timeLimit` *parameter is set to infinity, the single neighborhood diving algorithm finishes in a finite number of iterations and either returns a feasible solution of the input problem, or proves the infeasibility of the input problem.*

Proof. Let $\widetilde{x}^i$ be the reference solution at the beginning of the ith iteration, obtained by solving the MIP problem $\mathrm{MIP}(Q_i, [\overline{x}])$ and let $j \geq i + 1$. The problem Q_j contains all constraints from Q_{i+1}. If the algorithm has reached the jth iteration, it means that in the ith iteration feasible solution was not found and cut $\delta(\widetilde{x}^i, x) \geq \lceil \delta(\widetilde{x}^i, \overline{x}) \rceil$ (line 9 in Algorithm 38) was added

to Q_{i+1}. Hence, the problem MIP$(Q_j, [\overline{x}])$ contains $\delta(\widetilde{x}^i, x) \geq \lceil \delta(\widetilde{x}^i, \overline{x}) \rceil$. Furthermore, because $\lceil \delta(\widetilde{x}^i, \overline{x}) \rceil > 0$ (otherwise, $\widetilde{x}^i$ would be feasible and the algorithm would stop in the ith iteration), this implies that $\widetilde{x}^i(\mathcal{B}) \neq \widetilde{x}^j(\mathcal{B})$. Since this reasoning holds for any two iterations $j > i \geq 0$, the total number of iterations of the single neighborhood diving algorithm is limited by the number of possible sub vectors $\widetilde{x}^i(\mathcal{B})$, which is $2^{|\mathcal{B}|}$. Therefore, the single neighborhood diving algorithm finishes in a finite number of iterations.

The single neighborhood diving algorithm can only return a solution vector as a result if either $\lceil \delta(\widetilde{x}^i, \overline{x}) \rceil = 0$, therefore $\widetilde{x}^i$ being feasible for P, or if a feasible solution of the reduced problem $P(\widetilde{x}^i, J_k)$ is found. Since a feasible solution of $P(\widetilde{x}^i, J_k)$ is also feasible for P, this means that any solution vector returned by single neighborhood diving algorithm must be feasible for P.

Finally, we will prove that any feasible solution of P has to be feasible for Q_i, for any iteration $i \geq 0$. Moreover, we will prove that any feasible solution of P has to satisfy all constraints in Q_i, for any iteration $i \geq 0$. Since Q_0 does not contain any constraints, this statement is obviously true for $i = 0$. Let us assume that the statement is true for some $i \geq 0$, i.e. that for some $i \geq 0$ every feasible solution of P satisfies all constraints in Q_i. The problem Q_{i+1} is obtained from Q_i by adding constraints $\delta(\widetilde{x}^i, x) \geq \lceil \delta(\widetilde{x}^i, \overline{x}) \rceil$ and $\delta(J_k, \widetilde{x}^i, x) \geq 1$. According to the definition of $\lceil \delta(\widetilde{x}^i, \overline{x}) \rceil$, there cannot be any feasible solution of P satisfying the constraint $\delta(\widetilde{x}^i, x) < \lceil \delta(\widetilde{x}^i, \overline{x}) \rceil$. In other words, all feasible solutions of P must satisfy the constraint $\delta(\widetilde{x}^i, x) \geq \lceil \delta(\widetilde{x}^i, \overline{x}) \rceil$. Furthermore, if the constraint $\delta(J_k, \widetilde{x}^i, x) \geq 1)$ is added to Q_{i+1}, this means that the problem $P(\widetilde{x}^i, J_k) = (P \mid \delta(J_k, \widetilde{x}^i, x) = 0)$ is proven infeasible, and therefore no feasible solution of P can satisfy the constraint $\delta(J_k, \widetilde{x}^i, x) = 0$. Therefore, any feasible solution of P satisfies the constraints added to Q_i in order to obtain Q_{i+1} and hence any feasible solution of P satisfies all constraints in Q_{i+1}. This proves that any feasible solution of P satisfies all constraints in Q_i, for any $i \geq 0$. In other words, any feasible solution of P is feasible for Q_i, for any $i \geq 0$. Since MIP$(Q_{i+1}, [\overline{x}])$ has the same set of constraints as Q_i, this means that any feasible solution of P is feasible for MIP$(Q_i, [\overline{x}])$. As a consequence, if MIP$(Q_i, [\overline{x}])$ is proven infeasible for some $i \geq 0$, this implies that the original problem P is infeasible. $\square$

Extension to a General MIP Case

Obviously, fixing a certain number of variables can be performed for general MIP problems, as well as for 0-1 MIP problems. We here explain how the previously presented algorithms can be adapted and employed for solving the general MIP problems. In the case of VN diving, we compute the distances $\Delta_j =\mid \tilde{x}_j - \overline{x}_j \mid$, $j \in \mathcal{B} \cup \mathcal{G}$, for all integer variables (not just the binaries). Then, we successively solve subproblems $P(\tilde{x}, J_k)$, $J_k = \{1, \ldots, k\}$, $k =\mid \{j \in \mathcal{B} \cup \mathcal{G} : \widetilde{x}_j = \overline{x}_j\} \mid$, where $\widetilde{x}$ is the current reference integer solution and $\overline{x}$ is the solution of the LP relaxation of the original problem LP(P). If a feasible solution is found by solving $P(\tilde{x}, J_k)$, for some k, $0 \leq k \leq |\mathcal{B} \cup \mathcal{G}|$, it is returned as a feasible solution of the original problem P. In the VN diving variant for 0-1 MIP problems, a pseudo-cut is added to P if a subproblem $P(\tilde{x}, J_k)$ is proven infeasible. In the case of general MIP problems however, generating an appropriate pseudo-cut would require operating with extended problems, which contain significantly more variables and constraints than the original problem P. More precisely, the input problem would have to contain additional variables d_j, $j \in \mathcal{G}$, and additional constraints (see definition (3.5)):

$$u_j - d_j \leq x_j \leq d_j + l_j \;\text{ for all } j \in \{i \in \mathcal{G} \mid l_i < y_i < u_i\}.$$

Consequently, all subproblems derived from the so extended input problem would have to contain these additional variables and constraints. In order to save the memory consumption and computational time for solving subproblems, we therefore decide not to add any pseudo-cuts in the VN diving variant for general MIP problems, although that implies possible repetitions in the search space exploration. This means that we only perform decomposition with respect to the LP relaxation solution of the initial problem. In this aspect, VN diving for general MIP problems is similar to the Variable Neighborhood Decomposition Search (VNDS) algorithm for 0-1 MIP problems from [151].

In order to avoid memory and time consumption when dealing with large problems, the implementation of the SN diving algorithm for general MIP problems is the same as for 0-1 MIP problems. In other words, all distance values are computed with respect to the distance function δ (which takes into

account only binary variables), and general integer variables are handled by the generic MIP solver itself.

3.2.3 Computational Results

In this section we present the computational results for single and variable neighborhood diving algorithms. We compare our proposed methods with the following existing methods CPLEX MIP solver without feasibility pump (`CPLEX` for short), the standard feasibility pump heuristic (`standard FP`), the objective feasibility pump (`Objective FP`) and the variable neighborhood pump (`VNP`) [113]. Since the feasibility pump is already included as a primal heuristic in the employed version of the CPLEX MIP solver, we use the appropriate parameter settings to control the use of FP and to chose the version of FP. All results reported are obtained on a computer with a 4.5GHz Intel Core i7-2700K Quad-Core processor and 32GB RAM, using the general purpose MIP solver IBM ILOG CPLEX 12.4. Both algorithms were implemented in C++ and compiled within Microsoft Visual Studio 2010. For comparison purposes, we consider 83 0-1 MIP instances [77] previously used for testing the performance of the basic FP (see Table 3.1) and 34 general MIP instances previously used in [18] (see Table 3.2). In Tables 3.1 and 3.2, columns denoted by n represent the total number of variables, whereas columns denoted by $|\mathcal{B}|$ and m show the number of binary variables and the number of constraints, respectively. Additionally, the column denoted by $|\mathcal{G}|$ in Table 3.2 provides the number of general integer variables for a given instance.

In both proposed diving heuristics, the CPLEX MIP solver is used as a black-box for solving subproblems to feasibility. For this special purpose, the parameter `CPX_PARAM_MIP_EMPHASIS` is set to `FEASIBILITY`, the parameter `CPX_PARAM_INTSOLLIM` is set to 1 and the parameter `CPX_PARAM_FPHEUR` was set to -1. All other parameters are set to their default values, unless otherwise specified. Results for the CPLEX MIP solver without FP were obtained by setting the parameter `CPX_PARAM_FPHEUR` to -1. The feasibility pump heuristics are tested through the calls to the CPLEX MIP solver with the settings `CPX_PARAM_FPHEUR`=1 for standard FP and `CPX_PARAM_FPHEUR`=2 for

objective FP. All tested methods (CPLEX MIP without FP, standard FP, objective FP and both proposed diving heuristics) were allowed 100 seconds of total running time on 0-1 MIP test instances, while on General MIP instances maximum running time, for all methods, was set to 150 seconds. In addition, the time limit for solving subproblems within variable neighborhood diving and single neighborhood diving was set to 10 seconds for all instances.

The value of the neighborhood change control parameter d in the VN diving algorithm (see Algorithm 37) is set to 10, meaning that, in each iteration of VN diving, $10 + \lfloor 1 + \log_2(|\overline{x}_j \in \{0,1\} : j \in \mathcal{B}|) \rfloor$ subproblems (i.e. neighborhoods) are explored, where $\overline{x}$ is the LP relaxation solution of the current problem. The neighborhood size control parameter α in the SN diving algorithm (see Algorithm 38) is set to 2.5, meaning that $\frac{1}{2.5} \times 100 = 40$ percent of the variables with integer values in $\overline{x}$ are initially fixed to those values to obtain the first subproblem. Those values of d and α are based on brief experimental analysis.

No.	Instance name	n	$\lvert\mathcal{B}\rvert$	m	No.	Instance name	n	$\lvert\mathcal{B}\rvert$	m
1	10teams	2025	1800	230	43	bg512142	792	240	1307
2	aflow30a	842	421	479	44	dg012142	2080	640	6310
3	aflow40b	2728	1364	1442	45	blp-ar98	16021	15806	1128
4	air04	8904	8904	823	46	blp-ic97	9845	9753	923
5	air05	7195	7195	426	47	blp-ic98	13640	13550	717
6	cap6000	6000	6000	2176	48	blp-ir98	6097	6031	486
7	dano3mip	13873	552	3202	49	CMS750_4	11697	7196	16381
8	danoint	521	56	664	50	berlin_5_8_0	1083	794	1532
9	ds	67732	67732	656	51	railway_8_1_0	1796	1177	2527
10	fast0507	63009	63009	507	52	glass4	322	302	396
11	fiber	1298	1254	363	53	net12	14115	1603	14021
12	fixnet6	878	378	478	54	nsrand_ipx	6621	6620	735
13	harp2	2993	2993	112	55	tr12-30	1080	360	750
14	liu	1156	1089	2178	56	van	12481	192	27331
15	markshare1	62	50	6	57	biella1	7328	6110	1203
16	markshare2	74	60	7	58	NSR8K	38356	32040	6284
17	mas74	151	150	13	59	rail507	63019	63009	509
18	mas76	151	150	12	60	rail2536c	15293	15284	2539
19	misc07	260	259	212	61	rail2586c	13226	13215	2589
20	mkc	5325	5323	3411	62	rail4284c	21714	21705	4284
21	mod011	10958	96	4480	63	rail4872c	24656	24645	4875
22	modglob	422	98	291	64	A1C1S1	3648	192	3312
23	momentum1	5174	2349	42680	65	A2C1S1	3648	192	3312
24	nw04	87482	87482	36	66	B1C1S1	3872	288	3904
25	opt1217	769	768	64	67	B2C1S1	3872	288	3904
26	p2756	2756	2756	755	68	sp97ar	14101	14101	1761
27	pk1	86	55	45	69	sp97ic	12497	12497	1033
28	pp08a	240	64	136	70	sp98ar	15085	15085	1435
29	pp08aCUTS	240	64	246	71	sp98ic	10894	10894	825
30	protfold	1835	1835	2112	72	usAbbrv.8.25_70	2312	1681	3291
31	qiu	840	48	1192	73	manpower1	10565	10564	25199
32	rd-rplusc-21	622	457	125899	74	manpower2	10009	10008	23881
33	set1ch	712	240	492	75	manpower3	10009	10008	23915
34	seymour	1372	1372	4944	76	manpower3a	10009	10008	23865
35	swath	6805	6724	884	77	manpower4	10009	10008	23914
36	t1717	73885	73885	551	78	manpower4a	10009	10008	23866
37	vpm2	378	168	234	79	ljb2	771	681	1482
38	dc1c	10039	8380	1649	80	ljb7	4163	3920	8133
39	dc1l	37297	35638	1653	81	ljb9	4721	4460	9231
40	dolom1	11612	9720	1803	82	ljb10	5496	5196	10742
41	siena1	13741	11775	2220	83	ljb12	4913	4633	9596
42	trento1	7687	6415	1265					

Table 3.1: Benchmark instances for 0-1 MIP.

No.	**Instance name**	n	$\|\mathcal{B}\|$	$\|\mathcal{G}\|$	m
1	arki001	1388	415	123	1048
2	atlanta-ip	48738	46667	106	21732
3	gesa2	1224	240	168	1392
4	gesa2-o	1224	384	336	1248
5	ic97_potential	728	450	73	1046
6	ic97_tension	703	176	4	319
7	icir97_potential	2112	1235	422	3314
8	icir97_tension	2494	262	573	1203
9	manna81	3321	18	3303	6480
10	momentum2	3732	1808	1	24237
11	momentum3	13532	6598	1	56822
12	msc98-ip	21143	20237	53	15850
13	mzzv11	10240	9989	251	9499
14	mzzv42z	11717	11482	235	10460
15	neos7	1556	434	20	1994
16	neos8	23228	23224	4	46324
17	neos10	23489	23484	5	46793
18	neos16	377	336	41	1018
19	noswot	128	75	25	182
20	rococoB10-011000	4456	4320	136	1667
21	rococoB10-011001	4456	4320	136	1677
22	rococoB11-010000	12376	12210	166	3792
23	rococoB11-110001	12431	12265	166	8148
24	rococoB12-111111	9109	8910	199	8978
25	rococoC10-001000	3117	2993	124	1293
26	rococoC10-100001	5864	5740	124	7596
27	rococoC11-010100	12321	12155	166	4010
28	rococoC11-011100	6491	6325	166	2367
29	rococoC12-100000	17299	17112	187	21550
30	rococoC12-111100	8619	8432	187	10842
31	rout	556	300	15	291
32	timtab1	397	64	107	171
33	timtab2	675	113	181	294
34	roll3000	1166	246	492	2295

Table 3.2: Benchmark instances for general MIP.

No.	CPLEX	Standard FP	Objective FP	VNP	VN Diving	SN Diving	No.	CPLEX	Standard FP	Objective FP	VNP	VN Diving	SN Diving
1	952.00	952.00	964.00	**948.00**	994.00	956.00	43	9153557705.00	9153557705.00	9153557705.00	**120738665.00**	**120738665.00**	**120738665.00**
2	**1244.00**	**1244.00**	**1244.00**	1609.00	1342.00	1375.00	44	12028555539.00	12028555539.00	12028555539.00	**153406945.50**	**153406945.50**	**153406945.50**
3	1502.00	4703.00	2361.00	1663.00	**1370.00**	1973.00	45	**6910.11**	**6910.11**	**6910.11**	7684.11	7082.24	9434.80
4	57528.00	57528.00	57528.00	59145.00	**56635.00**	59834.00	46	**4656.92**	**4656.92**	**4656.92**	4745.99	4965.41	4934.99
5	29888.00	29888.00	29888.00	31988.00	31988.00	**29077.00**	47	**5042.91**	**5042.91**	**5042.91**	5493.79	10372.78	13698.32
6	-106024.00	-106024.00	-106024.00	-2445700.00	**-2451175.00**	-1318020.00	48	**2433.13**	3101.46	3294.20	2616.77	2845.44	3974.77
7	**727.22**	**727.22**	**727.22**	768.38	768.38	5000.00	49	993.00	993.00	993.00	**258.00**	358.00	335.00
8	**66.50**	79.00	**66.50**	83.00	77.00	69.50	50	100.00	100.00	100.00	70.00	73.00	**67.00**
9	5418.56	5418.56	5418.56	5418.56	**1573.84**	4004.71	51	500.00	500.00	500.00	416.00	413.00	**408.00**
10	733.00	733.00	733.00	**181.00**	**181.00**	585.00	52	3691696333.33	3691696333.33	3691696333.33	5200046900.00	**2850022250.00**	3066669866.67
11	451769.10	**436408.71**	752876.96	613482.32	1400807.72	608394.16	53	**214.00**	337.00	**214.00**	296.00	255.00	295.00
12	**4505.00**	**4505.00**	**4505.00**	**4505.00**	8621.00	6044.00	54	261440.00	261440.00	261440.00	261760.00	**195200.00**	225920.00
13	-44633925.00	-44633925.00	-44633925.00	-42637348.00	**-72614611.00**	-5404012.00	55	140249.00	285747.00	154248.00	197146.00	**134912.00**	137933.00
14	6450.00	6450.00	6450.00	**5762.00**	**5762.00**	**5762.00**	56	64.00	64.00	64.00	43.61	**4.82**	64.00
15	7286.00	7286.00	7286.00	**230.00**	**230.00**	**230.00**	57	45112270.55	45112270.55	45112270.55	**13439555.63**	**13439555.63**	11646076401
16	10512.00	10512.00	10512.00	**338.00**	**338.00**	**338.00**	58	5007115060.37	5007115060.37	5007115060.37	6336253949.24	6439612939.29	**2634420708.79**
17	157344.61	157344.61	157344.61	**14372.87**	**14372.87**	123739.83	59	451.00	451.00	451.00	**181.00**	**181.00**	398.00
18	157344.61	157344.61	157344.61	**43774.26**	**43774.26**	119400.61	60	1698.00	1698.00	1698.00	**711.00**	**711.00**	1889.00
19	3370.00	4365.00	3545.00	3620.00	**3330.00**	3375.00	61	2395.00	2395.00	2395.00	**1001.00**	**1001.00**	2348.00
20	0.00	0.00	0.00	0.00	**-261.75**	-92.21	62	2701.00	2701.00	2701.00	**1126.00**	**1126.00**	2805.00
21	0.00	0.00	0.00	0.00	0.00	0.00	63	3761.00	3761.00	3761.00	**1614.00**	**1614.00**	3400.00
22	36180511.32	36180511.32	36180511.32	**35147088.88**	**35147088.88**	**35147088.88**	64	19005.23	19005.23	15838.91	13081.23	**12259.60**	18962.51
23	**314555.10**	**314555.10**	**314555.10**	372338.73	372351.67	430397.54	65	20865.33	20865.33	20865.33	**11934.26**	12151.49	20625.65
24	**17004.00**	**17004.00**	**17004.00**	19792.00	19792.00	24228.00	66	69933.52	69933.52	69933.52	28798.94	**28034.72**	69933.52
25	0.00	0.00	0.00	**-16.00**	**-16.00**	0.00	67	70575.52	70575.52	70575.52	30885.28	**27474.62**	70375.52
26	3705.00	3705.00	3705.00	5728.00	**3131.00**	3711.00	68	1128647874.42	1128647874.42	1128647874.42	1128651978.42	**1149811855.24**	624182788302
27	731.00	731.00	731.00	**18.00**	**18.00**	**18.00**	69	14643093330.88	14643093330.88	14643093330.88	**747411597.12**	**747411597.12**	13125332908.88
28	27080.00	27080.00	27080.00	**9140.00**	10760.00	25160.00	70	2374928235.04	2374928235.04	2374928235.04	2349465005.92	**893040556.48**	4456259661.60
29	13690.00	13690.00	13690.00	**8250.00**	10940.00	11820.00	71	16956655079.52	16956655079.52	16956655079.52	**53550136736**	6582187990.04	13678053346.56
30	-13.00	-13.00	-13.00	-16.00	**-17.00**	-13.00	72	200.00	200.00	200.00	144.00	**131.00**	**131.00**
31	1805.18	1805.18	1805.18	385.91	**-1.21**	1577.11	73	**6.00**	9.00	**6.00**	**6.00**	7.00	**6.00**
32	186117.18	184040.51	184732.09	-	177132.88	**165894.84**	74	6.00	6.00	6.00	6.00	6.00	6.00
33	479735.75	479735.75	479735.75	389031.00	**77487.00**	235084.50	75	6.00	6.00	6.00	6.00	6.00	6.00
34	666.00	666.00	666.00	**473.00**	**473.00**	632.00	76	6.00	6.00	6.00	6.00	6.00	6.00
35	713.21	713.21	713.21	713.21	**512.09**	1955.96	77	**6.00**	**6.00**	**6.00**	**6.00**	7.00	**6.00**
36	258934.00	258934.00	258934.00	322075.00	320556.00	**218167.00**	78	6.00	6.00	6.00	6.00	6.00	6.00
37	17.00	17.00	17.00	15.75	19.75	**15.50**	79	**0.88**	**0.88**	**0.88**	7.24	7.24	7.24
38	1628753907.38	1628753907.38	1628753907.38	**15919302.95**	16271096.12	11691888636.35	80	**0.87**	**0.87**	**0.87**	8.61	8.61	8.61
39	16644728415.79	16644728415.79	16644728415.79	**26768702.57**	27048192.51	13567591658.00	81	**1.42**	**1.42**	**1.42**	9.48	9.48	9.48
40	2386640556.42	2386640556.42	2386640556.42	**19998521.16**	**19998521.16**	1866404347.13	82	**1.63**	**1.63**	**1.63**	7.31	7.31	7.31
41	560995341.87	560995341.87	560995341.87	**12115156.75**	**12115156.75**	327279193.17	83	**1.91**	**1.91**	**1.91**	6.20	6.20	6.20
42	57491611137.01	57491611137.01	57491611137.01	**429883219.07**	**429883219.07**	44387127711.00	*Wins*						

Table 3.3: Objective values for 0-1 MIP instances.

No.	CPLEX	Standard FP	Objective FP	VNP	VN Diving	SN Diving	No.	CPLEX	Standard FP	Objective FP	VNP	VN Diving	SN Diving
1	0.96	1.23	1.20	0.61	0.52	**0.40**	43	**0.01**	**0.01**	**0.01**	0.08	0.05	0.05
2	0.09	0.04	**0.02**	0.05	0.23	0.05	44	**0.01**	**0.01**	**0.01**	0.32	0.18	0.18
3	0.24	**0.07**	0.13	0.55	3.21	0.24	45	2.11	2.38	2.33	2.59	10.78	**1.10**
4	**0.48**	0.49	0.53	2.21	0.98	0.56	46	0.20	**0.19**	**0.19**	0.74	5.68	0.23
5	**0.14**	**0.14**	**0.14**	0.42	0.27	0.35	47	0.42	0.39	0.40	1.03	0.44	**0.19**
6	**0.06**	**0.06**	**0.06**	**0.06**	0.30	0.07	48	0.56	0.18	0.20	0.64	0.57	**0.12**
7	59.58	60.21	60.00	5.31	**3.53**	13.11	49	0.22	**0.20**	**0.20**	31.46	17.4	5.23
8	0.36	0.22	0.68	0.25	**0.05**	0.10	50	**0.01**	**0.01**	**0.01**	0.44	0.64	0.25
9	0.42	**0.41**	0.42	13.40	78.78	364.69	51	**0.02**	0.03	**0.02**	3.75	1.34	0.28
10	**0.26**	**0.26**	**0.26**	1.45	0.92	0.84	52	**0.01**	**0.01**	**0.01**	0.04	0.13	**0.01**
11	0.02	**0.01**	0.02	0.03	**0.01**	0.02	53	29.52	1.88	30.18	6.37	2.31	1.70
12	**0.01**	**0.01**	**0.01**	0.02	0.02	**0.01**	54	0.14	**0.11**	**0.11**	0.43	0.38	0.12
13	**0.01**	**0.01**	**0.01**	0.04	0.22	**0.01**	55	0.86	**0.04**	**0.04**	18.50	10.03	0.35
14	**0.01**	**0.01**	**0.01**	0.03	0.03	0.02	56	48.06	48.33	47.96	100.19	12.92	**8.03**
15	0.00	0.00	0.00	0.00	0.00	0.00	57	**0.04**	**0.04**	**0.04**	0.88	0.59	0.50
16	0.00	0.00	0.00	0.00	0.00	0.00	58	13.77	13.75	**13.73**	46.77	34.79	29.49
17	0.00	0.00	0.00	0.00	0.00	0.00	59	**0.32**	**0.32**	**0.32**	2.10	1.50	1.08
18	0.00	0.00	0.00	0.00	0.00	0.00	60	**0.10**	**0.10**	**0.10**	4.18	2.47	2.37
19	0.14	0.05	0.07	0.09	0.04	**0.01**	61	0.20	**0.19**	0.20	3.21	1.94	1.83
20	**0.04**	**0.04**	**0.04**	0.13	0.08	**0.04**	62	0.33	**0.32**	0.35	14.99	8.71	8.08
21	**0.04**	**0.04**	**0.04**	0.05	**0.04**	**0.04**	63	0.41	**0.40**	0.41	8.36	5.42	4.28
22	**0.00**	**0.00**	**0.00**	0.01	**0.00**	**0.00**	64	**0.13**	0.51	**0.13**	15.81	10.37	0.26
23	0.77	**0.75**	**0.75**	15.01	24.82	12.67	65	**0.06**	**0.06**	**0.06**	24.17	10.09	0.09
24	1.05	1.23	1.30	0.60	0.42	**0.41**	66	**0.06**	**0.06**	0.07	19.26	10.11	0.10
25	**0.00**	0.01	0.01	0.01	**0.00**	0.01	67	**0.07**	**0.07**	**0.07**	16.99	10.13	0.09
26	**0.04**	**0.04**	**0.04**	0.07	0.34	0.17	68	**0.36**	**0.36**	**0.36**	3.19	1.55	0.90
27	0.00	0.00	0.00	0.00	0.00	0.00	69	0.22	**0.21**	**0.21**	0.87	0.55	0.54
28	**0.00**	**0.00**	**0.00**	0.03	0.01	**0.00**	70	0.35	**0.34**	**0.34**	2.75	1.07	0.99
29	**0.01**	**0.01**	**0.01**	0.05	**0.01**	**0.01**	71	0.25	**0.24**	0.25	1.59	0.76	0.51
30	16.04	16.33	16.36	4.54	10.69	**2.86**	72	**0.03**	**0.03**	**0.03**	13.49	11.24	0.98
31	**0.04**	**0.04**	**0.04**	1.34	0.99	0.06	73	1.23	**0.92**	0.93	5.10	2.86	2.25
32	29.61	37.02	44.55	100.06	26.47	**4.89**	74	**5.15**	5.64	5.56	10.74	6.68	7.59
33	**0.01**	**0.01**	**0.01**	0.04	**0.01**	**0.01**	75	**3.94**	4.77	4.79	9.60	7.46	4.57
34	**0.04**	**0.04**	**0.04**	1.19	0.59	0.67	76	**4.60**	5.37	5.48	14.03	9.05	5.12
35	**0.06**	**0.06**	**0.06**	0.20	20.17	20.36	77	**3.13**	3.70	3.77	12.62	7.64	4.16
36	100.15	100.48	100.61	**8.85**	14.85	10.57	78	**5.09**	5.87	5.92	12.32	9.52	7.08
37	**0.01**	**0.01**	**0.01**	0.02	0.02	**0.01**	79	**0.01**	**0.01**	**0.01**	0.03	0.03	0.03
38	**0.07**	**0.07**	**0.07**	3.25	1.87	1.72	80	0.15	**0.14**	**0.14**	0.65	0.58	0.57
39	0.26	**0.25**	**0.25**	5.98	2.50	2.32	81	0.19	0.19	**0.17**	0.85	0.72	0.70
40	**0.10**	**0.10**	**0.10**	3.16	2.11	1.87	82	0.23	0.24	**0.22**	1.33	1.13	1.12
41	2.32	**2.31**	2.28	7.75	5.48	5.71	83	**0.18**	**0.18**	**0.18**	0.96	0.86	0.81
42	**0.04**	**0.04**	**0.04**	2.55	1.57	1.66	Avg.	4.05	3.85	4.29	7.14	5.09	6.63

Table 3.4: Execution time values (in seconds) for 0-1 MIP instances.

	CPLEX	Standard FP	Objective FP	VNP	VN Diving	SN Diving
Solution quality						
Instances solved	83	83	83	82	83	83
Avg. gap from LP relaxation obj. w.r.t. all instances (%)	49665.28	49666.96	49649.94	-	6620.55	17890.24
Avg. gap from LP relaxation obj. w.r.t. 82 instances solved by VNP (%)	48002.46	48029.48	48003.82	4683.57	4542.36	16086.52
Number of wins	18	15	17	32	44	17
Computational time						
Average w.r.t. all instances(sec)	4.05	3.85	4.29	7.14	5.09	6.63
Average w.r.t. 82 instances solved by VNP	3.74	3.45	3.79	6.01	4.83	6.65
Number of wins	42	50	48	2	8	19

Table 3.5: Summarized results for 0-1 MIP instances.

The results obtained by all 6 solvers, for the first 83 benchmark 0-1 MIP instances, which were first used in [77], are presented in Tables 3.3 and 3.4. Table 3.3 provides the objective values obtained by all 6 methods and Table 3.4 provides the corresponding execution time. The summarized results for this benchmark, including the variable neighborhood pump heuristic [113], are presented in Table 3.5. In the solution quality block of Table 3.5, we provide the number of instances solved by each of the 6 methods, the average percentage gap from the LP relaxation objective value regarding all 83 instances, the average percentage gap from the LP relaxation objective value regarding the instances solved by VNP, and the number of times that each of the methods managed to obtain the best objective value among the others (including ties). For each method, a percentage gap for a particular instance was computed according to the formula $\frac{f-f_{LP}}{|f_{LP}|} \times 100$, where f is the objective function value for the observed instance obtained by that method, and f_{LP} is the objective function value of the LP relaxation of the observed instance. The exceptions are instances `markshare1`, `markshare2` and `mod011`, for which the gap value was computed as $(f - f_{LP}) \times 100$, since the LP relaxation objective value is equal to 0 for all three instances. In the computational time block of the Table 3.5, we provide the average computational time over all instances in the benchmark for each of the 6 methods compared, the average computational time over all instances solved by VNP, as well as the number of times that each of the methods managed to obtain a solution in shortest time.

No.	Objective values						Running times						
	CPLEX	**Standard FP**	**Objective FP**	**VNP**	**VN Diving**	**SN Diving**	**No.**	**CPLEX**	**Standard FP**	**Objective FP**	**VNP**	**VN Diving**	**SN Diving**
1	7583912.00	7583912.00	7583912.00	**7583157.35**	7616973.21	7607607.29	1	0.50	0.87	0.84	1.44	10.2	**0.33**
2	106.01	106.01	106.01	-	**97.01**	128.01	2	122.88	123.42	124.53	150.08	45.31	**15.72**
3	83888091.14	83888091.14	83888091.14	166697234.63	**26334562.23**	68817832.42	3	0.02	**0.01**	0.02	0.06	0.04	0.02
4	**25792003.54**	29819721.89	26854088.34	**25792003.54**	26334562.23	25815849.31	4	0.04	0.04	0.05	0.09	0.04	**0.02**
5	4263.00	4206.00	4206.00	4275.00	**4042.00**	4279.00	5	**0.07**	0.42	0.47	20.38	1.52	0.12
6	4084.00	4077.00	4077.00	4086.00	4076.00	**4044.00**	6	0.17	0.52	0.54	2.54	0.73	**0.15**
7	7082.00	6966.00	6966.00	7088.00	**6449.00**	6845.00	7	1.03	0.69	0.65	31.05	10.28	**0.42**
8	6451.00	6470.00	6687.00	**6417.00**	6594.00	6630.00	8	4.67	6.17	**0.35**	22.01	11.18	1.58
9	0.00	0.00	0.00	**-6994.00**	**-6994.00**	**-6994.00**	9	**0.01**	**0.01**	**0.01**	0.11	0.06	0.06
10	**15218.97**	**15218.97**	**15218.97**	-	21216.59	25720.52	10	34.74	35.59	35.38	150.03	136.22	**13.88**
11	525000.00	525000.00	525000.00	525000.00	**325184.11**	496675.41	11	1.27	1.26	**1.25**	54.98	39.58	29.10
12	23307748.01	23307748.01	23307748.01	-	**22579326.01**	26502059.01	12	83.81	84.51	85.01	150.12	14.67	**4.98**
13	0.00	0.00	0.00	0.00	**-19078.00**	0.00	13	**0.71**	0.76	**0.71**	8.47	7.17	3.28
14	0.00	0.00	0.00	0.00	**-20370.00**	-9052.00	14	0.83	**0.81**	**0.81**	10.12	12.72	4.57
15	**723934.00**	5741899.00	853405.60	**723934.00**	4183899.00	**723934.00**	15	0.23	**0.07**	0.09	0.19	0.14	0.13
16	0.00	0.00	0.00	0.00	**-3719.00**	0.00	16	**0.14**	**0.14**	**0.14**	8.17	4.72	4.95
17	2.00	2.00	2.00	2.00	**-952.00**	2.00	17	0.14	0.13	**0.12**	3.00	2.75	1.99
18	447.00	447.00	451.00	**446.00**	450.00	450.00	18	13.11	11.58	11.95	8.95	13.79	**7.7**
19	**-40.00**	**-40.00**	**-40.00**	-34.00	-34.00	-34.00	19	0.01	0.01	0.01	**0.00**	**0.00**	**0.00**
20	55270.00	55270.00	55270.00	**31330.00**	52926.00	34068.00	20	0.17	0.18	0.18	0.85	**0.16**	0.19
21	59350.00	59350.00	59350.00	61176.00	52926.00	**37673.00**	21	0.17	0.18	0.18	0.60	**0.16**	0.41
22	99336.00	99336.00	99336.00	99336.00	96932.00	**96692.00**	22	0.36	0.38	0.39	1.31	**0.35**	0.50
23	99336.00	99336.00	99336.00	99336.00	99771.00	**99152.00**	23	0.70	0.75	0.72	2.17	**0.69**	0.95
24	55765.00	59139.00	56732.00	65252.00	**48062.00**	53453.00	24	50.89	36.91	32.63	11.74	11.03	**4.91**
25	26633.00	26633.00	26633.00	**17843.00**	23730.00	30062.00	25	0.09	**0.08**	**0.08**	0.33	**0.08**	0.09
26	34954.00	87563.00	**28095.00**	37521.00	69850.00	32054.00	26	2.92	0.38	0.88	1.26	**0.15**	0.25
27	146524.00	146524.00	146524.00	146524.00	**143170.00**	146524.00	27	0.47	0.40	0.40	1.27	**0.32**	0.37
28	146524.00	146524.00	146524.00	**83853.00**	128760.00	146524.00	28	0.24	0.24	0.23	0.77	0.22	**0.19**
29	**52189.00**	**52189.00**	**52189.00**	63480.00	117774.00	74202.00	29	2.91	3.19	3.29	3.92	0.74	**0.91**
30	102274.00	82848.00	82848.00	110131.00	115593.00	**47383.00**	30	0.73	1.08	1.08	1.61	**0.48**	0.63
31	2375.25	2375.25	2375.25	2375.25	**1231.38**	1383.98	31	**0.01**	**0.01**	**0.01**	0.05	0.18	0.08
32	1040909.00	986130.00	1114433.00	1115617.00	**846107.00**	1055386.00	32	0.16	0.12	**0.05**	0.69	1.12	0.08
33	1549355.00	1403843.00	1544127.00	1691420.00	**1231580.00**	1492526.00	33	0.72	2.63	2.7	0.82	10.05	**0.18**
34	15850.00	15850.00	15850.00	16259.00	**13095.00**	16049.00	34	0.74	1.05	1.01	1.56	2.58	**0.48**
Wins	5	3	4	9	17	7	*Avg.*	9.58	9.25	9.02	19.14	9.98	2.92

Table 3.6: Solution quality and running time performance for general MIP instances.

From Tables 3.3 and 3.5, we can see that all methods except VNP are able to solve all 83 instances. VNP does not manage to solve just one test instance. Therefore, the comparison of performances of CPLEX MIP solver, standard FP, objective FP, VN diving and SN diving has been done regarding all 83 instances, while the performances of VNP has been evaluated relatively to the performances of the previous five methods regarding 82 instances solved by VNP. It appears that VN diving clearly outperforms all other methods regarding the solution quality. Indeed, it manages to solve all 83 instances from the benchmark and has the smallest average gap (6620.55%) from the LP relaxation objective. In addition, VN diving provides the best objective values among all 6 methods in 44 out of 83 instances. That is much more than number of times that VNP (32 times), CPLEX MIP solver(18 times), objective FP (17 times) or standard FP(17 time) succeeds to reach best objective value. The second best among methods able to solve all 83 instances is SN diving with an average gap from the LP relaxation of 17890.24%. It is followed by objective FP (49649.94%), standard FP (49666.96 %) and CPLEX MIP solver (49665.28 %). On the other hand, regarding 82 instances solved by VNP, VNP has much smaller average gap from LP relaxation objective (4683.57%) in comparison with SN diving (16086.52%), CPLEX MIP solver (48002.46%), objective FP (48003.82 %) and standard FP (48029.48 %). However with respect to the average gap from LP relaxation objective, VNP is the second best method. Its average gap is slightly greater than the average gap of VN diving whose gap is 4542.36%.

From Tables 3.4 and 3.5, we can observe that the shortest average computational time of 3.85s is reported by standard FP, whereas objective FP and CPLEX MIP solver are only slightly slower with the average computational time of 4.29s and 4.05s, respectively. They are followed by VN diving, whose average computational time is 5.09s, whereas SN diving and VNP are the slowest, with 6.63s and 7.14s average computational time, respectively. Note, that in computation of average computational time of VNP, we include the time of its failed run. Also, note that on one instance (i.e., ds), we allowed to SN diving more than 100s of computational time and counted that run as successful. However, if we consider the average computational time of all six methods over all instances solved successfully by each of them (82 instances

	CPLEX	Standard FP	Objective FP	VNP	VN Diving	SN Diving
Solution quality						
Instances solved	34	34	34	31	34	34
Avg. gap from LP relaxation obj. w.r.t all instances(%)	403.44	454.18	407.46	-	383.28	381.08
Avg. gap from LP relaxation obj. w.r.t instances solved by VNP(%)	437.31	492.95	441.72	431.42	413.00	406.69
Number of wins	5	3	4	9	17	7
Computational time						
Average w.r.t all instances(sec)	9.58	9.25	9.02	19.14	9.98	2.92
Average w.r.t instances solved by VNP(sec)	2.72	2.29	1.99	6.47	4.62	2.09
Number of wins	5	7	10	1	9	14

Table 3.7: Summarized results for general MIP instances.

solved by VNP), the ranking of methods is almost unchanged besides that VNP is now faster than SN diving. Regarding the number of wins, the objective FP, the standard FP, and the CPLEX MIP manage to obtain a solution in the shortest time most often, in 50, 48 and 42 cases, respectively. The SN diving and VN diving follow, obtaining a solution in the shortest time in 19, and 8 cases, respectively. The VNP has the worst performance in this respect, since it finds a solution before other methods in just two cases.

The objective function values and the corresponding execution time for the second benchmark of 34 general MIP instances [18] are presented in Table 3.6. Summarized results for this benchmark are presented in Table 3.7. For each method, a percentage gap for a particular instance was computed according to the formula

$$\frac{f - f_{LP}}{|f_{LP}|} \times 100,$$

where f is the objective function value for the observed instance obtained by that method, and f_{LP} is the objective function value of the LP relaxation of the observed instance. Note that for this benchmark set, there is no exception to this rule since there is no instance whose LP objectives is equal to 0.

From Tables 3.6 and 3.7, we can see that again only the VNP is not able to solve all 34 instances. Therefore, the comparison of performances of CPLEX without FP, standard FP, objective FP, VN diving, and SN diving has been done in the same way as for the previous benchmark set. From Tables 3.6 and 3.7, we conclude that VN diving and SN diving have best performances regarding the solution quality. The SN diving heuristic achieves the smallest average gap from the LP objective (381.08%) and obtains the best objective among all 6 methods in 7 cases. The VN diving has a slightly worse average gap of 383.28%, but obtains the best objective among all methods in 17

cases. If we take into account the average computational time of these two methods, we may conclude that SN diving is the best method for the general MIP problem. The third best method appears to be the CPLEX MIP solver without FP, with 403.44% average LP relaxation gap and 5 wins, followed by objective FP with 407.46% average gap and 4 wins. The standard FP heuristic has a significantly higher gap from the LP relaxation (454.18 %) and only 3 objective wins, indicating that FP is the worst choice quality-wise for the general MIP benchmark. Moreover, the ranking of CPLEX without FP, standard FP, objective FP, VN diving, and SN diving regarding solution quality on instances solved by VNP is the same. However, on these instances, VNP manifests much better behavior than CPLEX without FP, standard FP, objective FP regarding the average gap from the LP value. Additionally, VNP has 9 objective wins, indicating that VNP is the second best method, after VN diving, regarding the number of wins.

From Tables 3.6 and 3.7, we can see that SN diving achieves the impressive average execution time of 2.92s. The next method, according to the average execution time, is the objective FP heuristic which is more than three times slower, with average computational time of 9.02s. It is followed by standard FP with 9.25s average time, the CPLEX MIP solver without FP with 9.58s average time, VN diving (9.98s), and finally the VNP heuristic, which is the slowest method with 19.14s average computational time. Moreover, the ranking of methods remains the same even in case that the average computational times are computed regarding instances solved by VNP. Regarding number of wins, the SN diving manages to obtain a solution in the shortest time in 14 cases. The objective FP, VN diving, standard FP, and CPLEX MIP solver follow by obtaining a solution in the shortest time in 10, 9, 7, 5 cases, respectively. The VNP has the worst performance, since it manages to find a solution before other methods in just one case.

According to the above experimental analysis, our two proposed diving heuristics generally provide solutions of a better quality than the CPLEX MIP solver and the two FP heuristics, within a similar or shorter computational time. Although the VNP heuristic proves to be highly competitive for the 0-1 MIP benchmark, it shows a rather poor performance for the general MIP benchmark. We may therefore claim that, in overall, VN diving heuristic and

SN diving outperform all four state-of-the-art solvers which were used for comparison purposes regarding solution quality. Additionally, we may claim that SN diving significantly outperforms all tested methods regarding average computational time needed to provide a feasible solution for the instances from General MIP benchmark.

3.2.4 Influence of the time limit on the performances of all six methods

In this section we check the imposed time limit influence on the number of solved instances by each method. The results are given in Table 3.8 and Figure 3.1 for 0-1 MIP instances, and Table 3.2 and Figure 3.2 for General MIP benchmark instances.

Time limit (s)	CPLEX	Standard FP	Objective FP	VNP	VN Diving	SN Diving
1	67	67	67	40	43	54
5	74	74	73	58	58	70
10	76	77	76	66	67	77
20	78	79	78	78	78	80
30	80	79	78	79	81	82
40	80	80	79	80	82	82
50	81	81	81	81	82	82
60	82	81	82	81	82	82
70	82	82	82	81	82	82
80	82	82	82	81	83	82
90	82	82	82	81	83	82
100	83	83	83	82	83	82

Table 3.8: Number of solved instances by 6 methods as a function of time limit - 0-1 MIP

It appears that CPLEX MIP solver, standard FP, and objective FP perform better if the time limit is less than 10s. However, increasing the time limit, the number of solved instances by the other methods grows dramatically. Consequently, when the time limit is set to 20 seconds, SN diving becomes the method with the most solved instances, keeping the first place until time limit is extended to 80 seconds, when VN diving becomes the best method able to solve all instances.

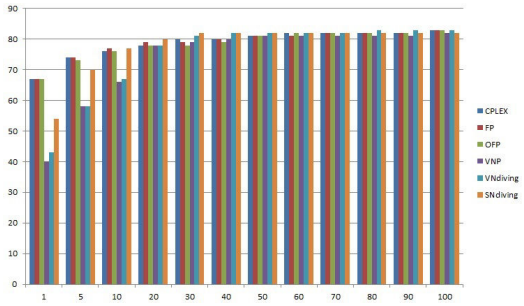

Figure 3.1: Number of solved instances by 6 methods as a function of time limit - 0-1 MIP

Time limit (s)	CPLEX	Standard FP	Objective FP	VNP	VN Diving	SN Diving
1	24	23	24	12	17	23
5	29	28	29	22	22	30
10	29	29	29	25	23	31
20	30	30	30	27	31	33
30	30	30	30	29	31	34
40	31	32	32	30	32	34
50	31	32	32	30	33	34
60	32	32	32	31	33	34
70	32	32	32	31	33	34
80	32	32	32	31	33	34
90	33	33	33	31	33	34
100	33	33	33	31	33	34
110	33	33	33	31	33	34
120	33	33	33	31	33	34
130	34	34	34	31	33	34
140	34	34	34	31	34	34
150	34	34	34	31	34	34

Table 3.9: Number of solved instances by 6 methods as a function of time limit - General MIP

From Table 3.9 and Figure 3.2, we conclude that CPLEX MIP solver, standard FP, objective FP, and SN diving are able to find a feasible solution within 1 second. The CPLEX MIP solver and objective FP manage to solve 24 instances out of 34 within 1 second, while standard FP and SN diving succeed to get 23 out of 34 instances in less than 1 second. Furthermore, it appears that SN diving outperforms all other methods if the time limits is greater than 1s. Moreover, SN diving solves all instances when the time limit is adjusted to 30 seconds; that is the smallest time limit that one method

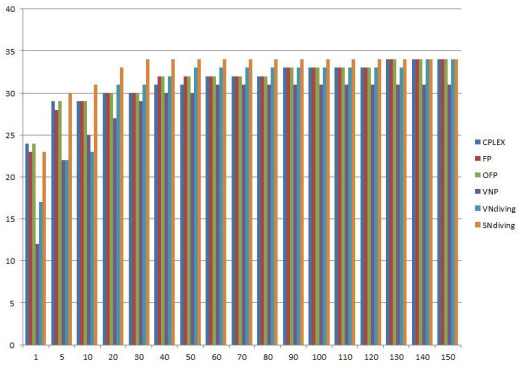

Figure 3.2: Number of solved instances by 6 methods as a function of time limit - General MIP

needs to solve all instances. Taking into account our previous observations, one can conclude that SN diving is the best heuristic for finding initial feasible solution for general MIP instances.

3.3 Efficient Matheuristics for Multicommodity Fixed-Charge Network Design

The multicommodity capacitated fixed-charge network design problem (MCND), is an NP-hard discrete optimization problem [159]. It is defined on a directed graph $G = (N, A)$, where N is the set of nodes and A is the set of arcs. Each commodity $k \in K$ has a demand $d_k > 0$ to be routed from an origin O_k to a destination D_k. Each arc (i, j) has the capacity $u_{ij} > 0$ on the flow of all commodities circulating on the arc, the fixed design cost $f_{ij} \geq 0$, and the transportation cost $c_{ij}^k \geq 0$ for commodity k. The problem consists of minimizing the total cost while satisfying the demands and respecting the capacity constraints. The total cost includes the total transportation cost for transferring commodities from the origins to the destinations and the total fixed cost for using arcs.

Let b_{ij}^k be equal to $\min\{d_k, u_{ij}\}$. Then, the MCND problem may be modeled as a mixed-integer program using continuous flow variables x_{ij}^k that represent the amount of flow on each arc (i, j) for each commodity k, and 0-1 design variables y_{ij} that indicate if the arc (i, j) is used or not:

$$\min v = f(x,y) = \sum_{k\in K}\sum_{(i,j)\in A} c_{ij}^k x_{ij}^k + \sum_{(i,j)\in A} f_{ij}y_{ij} \tag{3.12}$$

$$\sum_{j\in N_i^+} x_{ij}^k - \sum_{j\in N_i^-} x_{ji}^k = \begin{cases} d_k, & \text{if } i = O_k \\ 0, & \text{if } i \neq O_k, D_k, \quad \forall k \in K \\ -d_k, & \text{if } i = D_k \end{cases} \tag{3.13}$$

$$\sum_{k\in K} x_{ij}^k \leq u_{ij}y_{ij},\ \forall (i,j) \in A \tag{3.14}$$

$$x_{ij}^k \leq b_{ij}^k y_{ij},\ \forall (i,j) \in A, k \in K \tag{3.15}$$

$$x_{ij}^k \geq 0,\ \forall (i,j) \in A, k \in K \tag{3.16}$$

$$y_{ij} \in \{0,1\},\ \forall (i,j) \in A \tag{3.17}$$

where $N_i^+ = \{j \in N | (i,j) \in A\}$ is the set of successors of node i and $N_i^- = \{j \in N | (j,i) \in A\}$ is the set of predecessors of node i . Equations (3.13) are the flow conservation constraints for each node and each commodity. The capacity constraints (3.14) ensure that the capacity of each arc is respected. Additionally, they forbids any flow to circulate through an arc that is not chosen as part of the design. The so-called strong inequalities, (3.15), ensure the same, therefore, they are redundant. However, they significantly improve the linear programming (LP) relaxation bounds (Crainic et al. 1999 [48]). Note that model (3.12)-(3.17) represents the strong formulation of the MCND, while the weak formulation being obtained by removing constraints (3.15). Hence, their corresponding LP relaxations are, respectively, the strong and the weak relaxations.

For solving this NP-hard problem a plenty of exact and heuristic approaches have been proposed in the literature up to now. Regarding exact approaches a lot of work has been dedicated for developing Benders decomposition methods [44, 45], Lagrangian based procedures [48, 50, 78, 83, 130, 144, 201], branch-and-price and cutting plane methods [40, 84, 128]. On the other hand, regarding heuristic approaches able to produce high quality solutions there are: A slope scaling/Lagrangean perturbation heuristic [51], a simplex based tabu search proposed by Crainic et al. [49], heuristics that exploit cycle-based neighborhood [85, 86, 178, 179], a scatter search based heuristic [47], heuristics that use parallel cooperative strategies [46, 52], a capacity

scaling heuristic proposed by Katayama et al. [141]; a hybrid approach that combines simulated annealing and column generation techniques [226], local branching based heuristic [191], a hybrid approach that combines large neighborhood search and IP solver [127], a matheuristic combining an exact MIP method and a Tabu search metaheuristic [39], Capacity Scaling & Local Branching [140].

We propose several matheuristics for solving the multicommodity capacitated fixed-charge network design problem. The proposed matheuristics are based on adding pseudo-cuts in order to exclude a portion of solution space already examined, solving reduced problems deduced from the initial one and using heuristic to guide the search toward a near-optimal solution.

3.3.1 Slope scaling heuristic

The Slope Scaling (SS) [51] heuristic for the MCND is an iterative procedure that at each iteration solves a linear approximation of the original formulation. Note that similar ideas are used in Metaheuristic Search with Inequalities and Target Objectives [99, 100](see Chapter 1 of this thesis for more details). The objective coefficients in each linear approximation are adjusted so that the exact costs (both linear and fixed) incurred by the solution at the previous iteration are reflected. More precisely, the Slope scaling heuristic is based on solving a succession of linear multi-commodity minimum cost network flow problems, each defined by a vector of linearization factors $\rho(t)$ and denoted $MMCF(\rho(t))$:

$$\min \sum_{k \in K} \sum_{(i,j) \in A} (c_{ij}^k + \rho_{ij}^k(t)) x_{ij}^k \tag{3.18}$$

$$\sum_{j \in N_i^+} x_{ij}^k - \sum_{j \in N_i^-} x_{ji}^k = \begin{cases} d_k, & \text{if } i = O_k \\ 0, & \text{if } i \neq O_k, D_k, \\ -d_k, & \text{if } i = D_k \end{cases} \quad \forall k \in K \tag{3.19}$$

$$\sum_{k \in K} x_{ij}^k \leq u_{ij}, \ \forall (i,j) \in A \tag{3.20}$$

$$x_{ij}^k \geq 0, \ \forall (i,j) \in A, k \in K \tag{3.21}$$

When feasible or optimal solution x' of $MMCF(\rho(t))$ is known, a feasible solution of MCND may be derived by setting the design variables to

$$y'_{ij} = \begin{cases} 1, & \text{if } \sum_{k \in K} x'^k_{ij} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.22}$$

In that way one upper bound $v(t) = f(x', y')$ of a problem will be obtained. The values of $\rho(t+1)$ in the next iteration, i.e., $t+1$, is determined so as to reflect the exact costs in the iteration t. For example:

$$\rho_{ij}^k(t+1) = \begin{cases} f_{ij} / \sum_{k \in K} x'^k_{ij}, & \text{if } \sum_{k \in K} x'^k_{ij} > 0 \\ \rho_{ij}^k(t), & \text{otherwise} \end{cases} \tag{3.23}$$

The basic Slope Scaling heuristic works in the way presented at Algorithm 39. The Slope Scaling heuristic finishes its work as soon as the predefined stopping condition is met. The most common stopping criteria for a Slope Scaling heuristic are the following: The flows are repeated from the previous iteration; there is no progress on the best upper bound for β consecutive iterations; or a predefined number of iterations have been performed.

Algorithm 39: Slope Scaling heuristic

Function `SS()`;

1 $v^* = \infty, t = 0, \rho_{ij}^k(t) = f_{ij}/u_{ij}$;
2 **while** *stopping criterion is not satisfied* **do**
3 Solve $MMCF(\rho(t))$ to obtain x';
4 Compute upper bound $v(t) = f(x', y')$ using equation (3.22);
5 Update best value $v^* \leftarrow \min\{v^*, v(t)\}$;
6 Calculate $\rho(t+1)$ using rule (3.23);
7 $t \leftarrow t + 1$;

end

3.3.2 Convergent algorithm based on the LP-relaxation and pseudo-cuts

We develop several algorithms based on the LP-relaxation and pseudo-cuts for solving MCND problem. Those algorithms are based on ideas of solving (optimally or near optimally) a series of small sub-problems obtained from a series of linear programming relaxations within the search for an optimal solution of a given problem. The first algorithm of such type had been proposed in 1978 by Soyster et al. [204]. About twenty years later, Hanafi and Wilbaut revisited ideas of Soyster et al. and proposed several algorithms for solving the multidimensional knapsack problem [112, 221]. All these papers contain descriptions of exact algorithms along with proofs of their convergence toward to optimal solutions.

Formally, work of such an algorithm may be described in the following way. At each iteration, the LP-relaxation of the current MIP problem P is solved to generate one constraint. Then, a reduced problem induced from an optimal solution of the LP-relaxation is solved to obtain a feasible solution for the initial problem. If the stopping criterion is satisfied, then the best lower bound and the best upper bound are returned. Otherwise, a pseudo cut is added to P and the process is repeated.

More precisely let P denotes the 0-1 MIP problem we want to solve:

$$(P)\ \min\{cx + fy : Ax + By \leq b, y_j \in \{0,1\}, j \in N; x_k \in \mathbb{R}, k \in M\} \quad (3.24)$$

where $|N| = n$ and $|M| = m$. Then its *reduced problem* with respect to a binary solution $y' \in \{0,1\}^n$ and a subset $J \subseteq N$ is defined as:

$$P(y', J)\ \min\{cx + fy : Ax + By \leq b, y \in \{0,1\}^n, y_j = y'_j, j \in J, x \in \mathbb{R}^m\}. \quad (3.25)$$

Note that throughout the rest of the section, we assume that when y values are known, the values of x variables will be easily obtained solving LP problem. In other words, we assume that for a given y values, the values of x variables are uniquely determined.

Additionally, let P be an optimization problem and Q be a set of constraints. The notation $(P|Q)$ corresponds to the optimization problem ob-

tained from P by adding the set of constraints Q. The optimal value of the optimization problem P will be denoted by $v(P)$.

The *partial Hamming distance* between two binary solutions y and y' in $\{0,1\}^n$ relative to a subset $J \subset N$ is defined as:

$$\delta(y, y', J) = \sum_{j \in J} |y_j - y'_j|.$$

This distance $\delta(y, y', J)$ can be rewritten as follows:

$$\delta(y, y', J) = \sum_{j \in J} y_j(1 - y'_j) + y'_j(1 - y_j). \tag{3.26}$$

Note that the Hamming distance is equal to $\delta(y, y', N)$.

The main idea of the exact algorithm is adding a $pseudo - cut$ at each iteration in order to eliminate reduced problems already examined in the previous solution process. A pseudo-cut consists of linear inequality that excludes certain solutions from being feasible as solutions of the considered problem and it may not be valid in the sense of guaranteeing that at least one globally optimal solution will be retained in the feasible set.

The pseudo-code of a convergent algorithm based on the LP-relaxation and pseudo-cuts (CALPPC) is given in Algorithm 40. At each iteration, the LP-relaxation of the current problem Q is solved to obtain a lower bound $\underline{v}$ and its corresponding optimal solution $\bar{y}$. After that the reduced problem $P(\bar{y}, J)$, defined by chosen subset $J \subset N$, is solved to obtain an upper bound and the best upper bound is updated. Next, the current problem Q is updated by adding the pseudo-cut $\delta(y, \bar{y}, J) \geq 1$ in order to cut off the region explored by solving the reduced problem. The algorithm stops if the required tolerance

between the upper and the lower bounds is reached.

Algorithm 40: CALPPC

Function `CALPPC(P)`;
$Q = P$, $v^* = +\infty$;
repeat
  if Q infeasible **then break**;
  Solve the LP-relaxation of Q to obtain an optimal solution $\bar{y}$ and its objective value $\underline{v}$;
  Choose subset $J \subseteq N$;
  Solve the reduced problem $P(\bar{y}, J)$;
  Update the best known-value: $v^* = \min\{v^*, v(P(\bar{y}, J))\}$;
  Update the current problem Q by adding the pseudo-cut: $Q = (Q|\{\delta(y, \bar{y}, J) \geq 1\})$;
until $v^* - \underline{v} < \epsilon$;

Obviously, the work of such one algorithm relies on the fact how the set J is chosen since it determines the size of reduced problems to solve, as well as the size of region to be cut-off. Here, we propose two ways for choosing this set relatively to the optimal solution $\bar{y}$ of LP-relaxation Q:

- $J = J^0(\bar{y}) = \{j \in N : \bar{y}_j = 0\}$,
- $J = J^0(\bar{y}) \cup J^1(\bar{y})$ where $J^1(\bar{y}) = \{j \in N : \bar{y}_j = 1\}$,

Note that, if the set J is chosen in the first way, the resulting reduced problem is harder to solve than the one obtained choosing the set J in the second way. On the other hand, the part of the solution space, cut-off in the first case is larger than the part cut-off in the second case.

In what follows, we will show the convergence of Algorithm 40 in the case $J = J^0(\bar{y}) \cup J^1(\bar{y})$. The convergence of Algorithm 40 in the case $J = J^0(\bar{y})$ will not be presented here since it may be proven analogously to the case $J = J^0(\bar{y}) \cup J^1(\bar{y})$.

Proposition 3.3.1 *Let y' be a vector in $[0,1]^n$, the following inequality*

$$\sum_{j \in J^0(y')} y_j + \sum_{j \in J^1(y')} (1 - y_j) \geq 1 \tag{3.27}$$

cuts off any solution y'' dominated by the solution y', i.e., solution y'' such that $(J^0(y') \cup J^1(y')) \subseteq (J^0(y'') \cup J^1(y''))$.

Proof. The proof is trivial from the definition (3.26) of the partial Hamming distance, since for any solution y not dominated by y' we have $\delta(y, y', J^0(y') \cup J^1(y')) = 0$. □

Note that when the solution y' is a binary vector, the inequality (3.27) cuts off solution y' without cutting off any other solution in $\{0,1\}^n$. This pseudo cut is called *canonical cut* [6] on the unit hypercube $K = \{x \in \mathbb{R}^n : 0 \leq x_j \leq 1, j = 1, \ldots, n\}$.

Proposition 3.3.2 *Given a 0-1 MIP problem P. Let y' be an optimal solution of the LP-relaxation of P and y'' be an optimal solution for the reduced problem $P(y', J^0(y') \cup J^1(y'))$. Then, an optimal solution for P is either the feasible solution y'' or an optimal solution for the problem*

$$(P|\{\delta(y, y', J^0(y') \cup J^1(y')) \geq 1\}). \tag{3.28}$$

Proof. Let Y be the set of feasible solutions of the problem P. Then this set can be partitioned as $Y = (Y \cap \delta(y, y', J^0(y') \cup J^1(y')) = 0) \cup (Y \cap \delta(y, y', J^0(y') \cup J^1(y')) \geq 1)$. On the other hand, the set of feasible solutions of the reduced problem $P(y', J^0(y') \cup J^1(y'))$ is defined as $Y \cap \delta(y, y', J^0(y') \cup J^1(y')) = 0$, since the fixation constraints $y_j = y'_j$ for all $j \in J^0(y') \cup J^1(y')$ are equivalent to the constraint $\delta(y, y', J^0(y') \cup J^1(y')) = 0$. So, an optimal solution for P is either the optimal solution y'' of the reduced problem $P(y', J^0(y') \cup J^1(y'))$ or an optimal solution of the problem $P|\{\delta(y, y', J^0(y') \cup J^1(y')) \geq 1\}$. □

The following theorem states that, when the CALPPC algorithm terminates, the best solution found is an optimal solution for the initial problem.

Proposition 3.3.3 *Given a 0-1 MIP problem P, the algorithm CALPPC(P) returns an optimal solution for the problem P or indicates that the problem is infeasible in a finite number of iterations bounded by 3^n.*

Proof. Let $\overline{Y}$ be the feasible set of the LP-relaxation of the problem P associated to Y, the set of feasible solutions of the problem P. It is obvious that at each iteration, CALPPC(P) excludes the current optimal solution

of the LP-relaxation by adding the pseudo cut. Moreover, if the excluded part contains binary solutions, all these solutions had been already examined by solving the reduced problem. Since at each iteration we cut off a non empty part of $\overline{Y}$, and keep the best fond solution for initial problem, the CALPPC(P) will return an optimal solution or prove infeasibility of the problem.

On the other hand, for each solution $y' \in [0,1]^n$, we may associate a partial solution $y'' \in \{0,1,*\}^n$ such that

$$y''_j = \begin{cases} y'_j, & \text{if } y'_j \in \{0,1\} \\ *, & \text{if } y'_j \in]0,1[\end{cases} \tag{3.29}$$

Hence, at each iteration, CALPPC(P) solves a reduced problem $P(y'', J^0(y'') \cup J^1(y''))$ induced by a partial solution y'' associated to the fractional solution y' (i.e., an optimal solution of the current LP-relaxation). In that way, at each iteration exploration of the space defined by $Y \cap \delta(y, y', J^0(y') \cup J^1(y')) = 0$ is performed. However, according to the Proposition 3.3.2, by adding the pseudo-cut $\delta(y, y', J^0(y') \cup J^1(y')) \geq 1$ we avoid generation of the partial solution y'' in the further iterations of CALPPC. So, at each iteration we solve a reduced problem associated to a different partial solution. Hence, the number of iterations is limited by the number of different partial solutions, i.e., $3^n = |\{0,1,*\}^n|$. $\square$

Firstly, we test CALPPC algorithm in which a pseudo cut and a reduced problem are defined setting $J = J^0(\bar{y}) \cup J^1(\bar{y})$ on the test instance c33 ($|N| = 20, |A| = 230, |K| = 40$) in order to evaluate its convergence (see Section 3.3.4 for a description of the computer's characteristics and the description of instances). The reduced problems within CALPPC algorithm are solved using CPLEX 12.6 MIP solver. Since CALPPC algorithm required 152 iterations to converge, in Table 3.10, we report for some iterations the value of the LP relaxation (Column `Lower Bound`); the value of the feasible solution generated solving a reduced problem (Column `Upper Bound`) and the number of fractional variables in the solution of the LP relaxation which corresponds to the size of the reduced problem (Column `Size`). We observe that optimal solution of the problem is found in the third iteration, but

no. iteration	Lower bound	Upper bound	Size
1	422853.26	424385	14
2	422874.19	424385	18
3	422881.57	423848	16
4	422917.23	423848	20
...	...	...	...
149	423835.47	423848	51
150	423840.90	423848	49
151	423843.80	423848	50
152	423847.26	425378	20

Table 3.10: Example

CALPPC algorithm needed 152 iterations to prove its optimality. So, we may observe that the convergence is not easily reachable in practice. Additionally, the total running time of the algorithm was about 36 seconds while an optimal solution was found for 0.67 seconds. On the other hand CPLEX 12.6 MIP solver required only 0.60 seconds to solve this problem to optimality. This example seems to indicate that the CALPPC algorithm can not be used easily as an exact method. We may also conclude that the size of the reduced problem does not necessarily increase between two iterations as well as that the CALPPC algorithm could generate the same feasible solution several times. However, it seems that a CALPPC algorithm is able to produce good lower bounds in a small number of iterations.

The same conclusions are drawn when the CALPPC algorithm, where a pseudo cut and a reduced problem are defined setting $J = J^0(\bar{y})$, is run on the instance c33. CPLEX 12.6 MIP solver is again used for solving reduced problems within the CALPPC algorithm. Now, the CALPPC algorithm needed 0.59 seconds to reach an optimal solution, but 27.49 seconds to prove its optimality (or in total 126 iterations). All in all, the convergence of both algorithms is not easily reachable in practice.

3.3.3 Iterative linear programming-based heuristic

Due to the slow convergence of a CALPPC algorithm it cannot be used as an exact algorithm for large instances in practice. In that case, it is preferable to use it as a heuristic approach, imposing some other stopping criterion instead of requiring proof of optimality. The motivation for proposing heuristics based on the CALPPC algorithm stem from the fact that for many

instances CALPPC algorithm finds an optimal solution quickly but it needs a lot of CPU time to prove its optimality, as it was shown in the example 3.10. Additionally, sometimes solving reduced problems optimally is time consuming process itself and therefore it is better to use a heuristic approach for that purposes. All heuristic approaches derived from CALPPC framework we will refer as Iterative Linear Programming-based Heuristic (**ILPH**).

For example, a naive ILPH approach can be obtained by just stopping the CALPPC algorithm after certain number of iterations or after reaching the predefined CPU time limit. Moreover, the difficulty of resulting reduced problem required to solve in each iteration has big impact on the overall solution process. Hence, sometimes it is more beneficial to impose a CPU time limit for solving a reduced problem instead of solving it to optimality or to additionally reduce size of the reduced problem. The later is especially true in the case when the set $J^0(\bar{y})$ serves as set J, since the cardinality of the set $J^0(\bar{y})$ may be too small.

In this section, we propose how to use a slope scaling heuristic in order to speed up the process of solving a reduced problem as well as how to use it to detect additional variables that may be fixed in a reduced problem. Before we give more details how a slope scaling heuristic is used for these purposes, we show how we adapt a slope scaling heuristic to tackle reduced problems.

The adaptation is accomplished changing the way of choosing initial ρ values as well as changing the way of their updating during the solution process. More precisely, let $\bar{y}$ be the partial $0-1$ solution where some variables are fixed to 0 or 1 and other variables are free, i.e., $\forall(i,j) \in A,\ \bar{y}_{ij} \in \{0,1,*\}$. Assume further that we want to solve reduced problem obtained fixing all variables in $J^0(\bar{y})$ and $J^1(\bar{y})$. In order to forbid arcs (i,j) with $\bar{y}_{ij} = 0$, but use (favor) arcs (i,j) with $\bar{y}_{ij} = 1$ we determine $\rho_{ij}^k(t)$ as:

$$\rho_{ij}^k(t) = \begin{cases} \infty, & \text{if } \bar{y}_{ij} = 0 \\ 0, & \text{if } \bar{y}_{ij} = 1 \end{cases} \tag{3.30}$$

On the other hand, for all arcs $(i,j) \in A$ such that $\bar{y}_{ij} \in]0,1[$ we use the standard way for choosing initial ρ values as well as the standard updating

formulas. Namely, for each $\bar{y}_{ij} \in]0,1[$ we set:

$$\rho_{ij}^k(0) = f_{ij}/u_{ij}, \tag{3.31}$$

$$\rho_{ij}^k(t+1) = \begin{cases} f_{ij}/\sum_{k\in K} x'^k_{ij}, & \text{if } \sum_{k\in K} x'^k_{ij} > 0 \\ \rho_{ij}^k(t), & \text{otherwise} \end{cases} \tag{3.32}$$

The steps of the modified slope scaling heuristic are depicted at Algorithm 41.

Algorithm 41: Slope Scaling heuristic for solving reduced problem of MCND defined as $P(\bar{y}, J^0(\bar{y}) \cup J^1(\bar{y}))$

Function $SS\_RP\_1()$;
1 $v^* = \infty, t = 0$;
2 Determine $\rho(0)$ using formulas (3.30) and (3.31);
3 **while** *stopping criterion is not satisfied* **do**
4 Solve $MMCF(\rho(t))$ to obtain x';
5 Compute upper bound $v(t) = f(x', y')$ using equation (3.22);
6 Update best value $v^* = \min\{v^*, v(t)\}$;
7 Calculate $\rho(t+1)$ using equations (3.30) and (3.32);
8 $t \leftarrow t + 1$;
end

In the case that we want to solve a reduced problem obtained fixing variables in $J^0(\bar{y})$, we can use the previous heuristic, treating the set $J^1(\bar{y})$ as the empty set. Namely, it suffices to use the previously described way of forbidding arcs (i, j) with $\bar{y}_{ij} = 0$, while for all the other arcs use the standard way of determining their ρ values in each iteration.

The steps of a generic ILPH heuristic are given at Algorithm 42. The parameters t_{red} and t_{max} in Algorithm 42 represent the time limit for solving a reduced problem by the CPLEX 12.6 MIP solver and the time limit for overall procedure, respectively. In the pseudo-code the statement of the form $\texttt{CPLEX\_MIP}(\texttt{P}(\bar{\texttt{y}}, \texttt{J}'), \texttt{z}, \texttt{t}_{\texttt{red}})$ refers to a call to a generic MIP solver, to solve the reduced problem $P(\bar{y}, J')$ within t_{red} seconds, using a binary solution z as the starting one (Note that sometimes we will set $z = \emptyset$ meaning that no solution is provided to CPLEX 12.6 MIP.) Additionally, the statement $SS\_RP(J'', \bar{y})$ means that the modified Slope Scaling heuristic (explained above) is used to solve the reduced problem $P(\bar{y}, J'')$. From this scheme

seven heuristics described hereafter are deduced.

Algorithm 42: Steps of ILPH heuristic

Function $ILPH(P, t_{red}, t_{max})$;
1 $Q = P$;
2 **repeat**
3 **if** Q infeasible **then break**;
4 Solve the LP-relaxation of Q to obtain an optimal solution $\bar{y}$;
5 $y' \leftarrow SS\_RP(J'', \bar{y})$; //Optional step
6 $y'' \leftarrow CPLEX\_MIP(P(\bar{y}, J'), z, t_{red})$;
7 **if** y'' better than y^* **then** $y^* = y''$ // Update the best known-solution ;
8 Update the current problem Q by adding the pseudo-cut:
$Q = (Q|\{\delta(y, \bar{y}, J(\bar{y})) \geq 1\})$;
9 $t \leftarrow$ `CpuTime()`;
until $t > t_{max}$;

We propose three ILPH heuristics for MCND that use the set $J^0(\bar{y}) \cup J^1(\bar{y})$ to define a pseudo-cut and a reduced problem (i.e., in Algorithm 42, $J = J' = J'' = J^0(\bar{y}) \cup J^1(\bar{y})$). The main difference among them is the way in which a reduced 0-1 MIP problem is tackled. The first ILPH heuristic named ILPH_CPLEX uses the CPLEX 12.6 MIP solver (i.e., in Algorithm 42, Step 5 is discarded and $z = \emptyset$), the second one called ILPH_SS uses just the Slope Scaling heuristic to solve a reduced problem (i.e., in Algorithm 42, Step 6 is discarded), while the third one called ILPH_SSimp combines the Slope Scaling heuristic and CPLEX 12.6 MIP solver to solve a reduced problem (i.e., in Algorithm 42, $z = y'$). Namely, in order to solve a reduced problem ILPH_SSimp firstly applies the Slope Scaling heuristic to provide a good quality solution and after that applies CPLEX 12.6 MIP solver to improve further the obtained solution.

Additionally, we propose four ILPH heuristics that use the set $J^0(\bar{y})$ to define a pseudo-cut (i.e., in Algorithm 42, $J = J^0(\bar{y})$):

- ILPH_0_0 - In each iteration, the reduced problem defined as $P(\bar{y}, J^0(\bar{y}))$ and in order to solve it CPLEX 12.6 MIP is used (i.e., in Algorithm 42, $J' = J^0(\bar{y})$, Step 5 is discarded and $z = \emptyset$).

- ILPH_0_01 - In each iteration uses CPLEX 12.6 MIP to solve the reduced problem defined as $P(\bar{y}, J^0(\bar{y}) \cup J^1(\bar{y}))$ (i.e., in Algorithm 42,

$J' = J^0(\bar{y}) \cup J^1(\bar{y})$, Step 5 is discarded and $z = \emptyset$).

- ILPH_SS_0_0 - In each iteration, the reduced problem defined as $P(\bar{y}, J^0(\bar{y}))$. In order to solve the reduced problem, we firstly apply the modified Slope Scaling heuristic. The solution, returned by it is improved further putting back real cost to all open arcs. The values of variables in such obtained solution y' is compared with values of variables in LP solution $\bar{y}$. Each binary variable, that takes same, integer value, in solutions y' and $\bar{y}$ is fixed to that value in the reduced problem. After that the resulting reduced problem is solved employing CPLEX 12.6 MIP solver. The steps of this heuristic may be derived from Algorithm 42 setting $J'' = J^0(\bar{y})$, $J' = \{j \in N : \bar{y}_j = y'_j, \bar{y}_j \in \{0, 1\}\}$ and $z = \emptyset$.

- ILPH_SS_0_0_imp. - The only difference between this heuristic and ILPH_SS_zeros_zeros is that the improved solution returned by the slope scaling heuristic is provided as the starting solution for CPLEX 12.6 MIP solver when applied to solve a reduced problem. In other words, the only difference between them is that ILPH_SS_0_0_imp passes a solution y' of the modified Slope Scaling heuristic to CPLEX 12.6 MIP solver (i.e., in Algorithm 42 $z = y'$) unlike ILPH_SS_0_0 which invokes CPLEX 12.6 MIP solver in Algorithm 42 by $z = \emptyset$.

3.3.4 Computational results

All testing described in this section have been performed on a computer with Intel i7-4900MQ CPU 2.80 GHz and 16GB RAM. For testing purposes C and C+ benchmark instances described in Crainic et al.[50] have been used. These instances consist of general transshipment networks with one commodity per origin-destination and no parallel arcs. Each test instance is characterized by the number of nodes $n = |N|$, the number of commodities $p = |K|$, the number of arcs $m = |A|$, the degree of capacity tightness, with regard to the total demand, and importance of the fixed design cost, with respect to the transportation cost. The number of nodes n in an instance takes value of 25, 30 or 100; the number of commodities p vary from 10 to 400 commodities while the number of edges m vary from 100 to 700 arcs. All these instances

have been widely used in the literature and they are publicly available at `http://www.di.unipi.it/~frangio`.

When CPLEX 12.6 MIP solver is invoked within an ILPH heuristic, it is allowed to consume maximally 300 seconds (i.e., t_{red} is set to 300 seconds). On the other hand, when a Slope Scaling heuristic is used, it is terminated as soon as the objective function values do not change into two consecutive iterations. As stopping criterion for each of tested ILPH heuristic we set the time limit of 3600 seconds (i.e., t_{max} is set to 3600 seconds).

Testing of different Slope Scaling variants

In this section we present results obtained testing the following heuristic based on Slope Scaling:

- Basic Slope Scaling (`Basic SS`) heuristic presented in Section 3.3.1.
- Slope Scaling heuristic that include intensification phase (`SS_intens`). This heuristic is obtained, running Basic Slope Scaling and keeping L best encountered solutions during its execution. After that each of these L solutions is tried to be further improved, solving MMCF problem obtained putting high (infinity) cost for all arcs closed in the considered solution, and original cost for all the other arcs.
- Iterated Slope Scaling (`iterated_SS`). This heuristic is actually enhancement of `SS_intens` obtained iterating it, until there is no improvement yielded by the intensification step.

Basic Slope Scaling heuristic, either run as a stand-alone procedure or within another Slop Scaling based heuristic, finishes its work when total of m iterations is performed, while the size of the list containing the best encountered solutions, i.e. L is set to n.

The results of Slope Scaling variants are presented in Table 3.11. For each variant we report the value of the best found solution and total running time on a certain instance. Additionally, in columns '`SS_intens` (%) imp ' and '`iterated_SS` (%)imp' we report the percentage improvements, regarding solution value, attained by `SS_intens` over `Basic SS` and `iterated_SS` over `SS_intens`, respectively.

	Basic SS		SS_intens			iterated_SS		
Problem	value	time	value	time	(%)imp	value	time	(%)imp
100/400/010/F/L	28942.0	4.50	28905.0	5.77	0.13	28079.0	15.07	2.94
100/400/010/F/T	80372.0	5.04	78262.0	7.53	2.70	78118.0	20.55	0.18
100/400/010/V/L	28495.0	3.85	28487.0	3.98	0.03	28423.0	7.72	0.23
100/400/030/F/L	64395.0	8.48	59415.0	11.33	8.38	59308.0	30.41	0.18
100/400/030/F/T	155519.0	10.99	154311.8	28.26	0.78	153665.0	60.77	0.42
100/400/030/V/T	385161.0	7.70	385083.0	8.16	0.02	385083.0	14.06	0.00
20/230/040/V/L	426020.0	3.02	426020.0	3.25	0.00	426020.0	2.97	0.00
20/230/040/V/T	373657.0	3.11	372225.0	3.33	0.38	371642.0	10.03	0.16
20/230/040/F/T	655596.0	3.03	653849.0	3.38	0.27	653755.0	10.13	0.01
20/230/200/V/L	110001.0	17.72	104110.0	20.23	5.66	103206.0	47.24	0.88
20/230/200/F/L	154458.3	17.93	148431.0	20.39	4.06	147904.0	45.81	0.36
20/230/200/V/T	114524.0	14.46	109226.0	16.82	4.85	104729.0	55.37	4.29
20/230/200/F/T	154912.0	24.48	149150.2	30.08	3.86	146118.4	101.34	2.07
20/300/040/V/L	430373.0	4.58	430373.0	4.69	0.00	430373.0	4.48	0.00
20/300/040/F/L	606401.0	4.64	599301.0	5.06	1.18	596516.0	15.05	0.47
20/300/040/V/T	465647.0	4.69	464550.0	5.04	0.24	464550.0	9.75	0.00
20/300/040/F/T	618297.0	4.72	615468.0	4.96	0.46	614032.0	17.68	0.23
20/300/200/V/L	84955.0	26.12	83045.0	31.14	2.30	82279.0	73.58	0.93
20/300/200/F/L	138668.3	33.23	130832.0	39.64	5.99	128799.3	119.30	1.58
20/300/200/V/T	83390.0	23.67	81551.0	27.82	2.26	81275.0	74.66	0.34
20/300/200/F/T	119642.0	38.63	116859.8	46.48	2.38	116347.0	104.85	0.44
30/520/100/V/L	59222.0	25.30	57311.0	27.46	3.33	56816.0	105.70	0.87
30/520/100/F/L	113709.0	29.91	108475.0	32.65	4.83	106134.0	144.13	2.21
30/520/100/V/T	54043.0	25.91	53421.0	28.39	1.16	53254.0	114.28	0.31
30/520/100/F/T	107627.0	26.40	106399.0	30.37	1.15	104924.0	131.91	1.41
30/520/400/V/L	122475.0	185.77	118679.0	308.55	3.20	118371.2	730.14	0.26
30/520/400/F/L	168977.4	214.24	161824.0	467.05	4.42	161824.0	688.54	0.00
30/520/400/V/T	122816.4	134.09	119899.3	250.19	2.43	119089.2	1096.68	0.68
30/520/400/F/T	171619.0	243.78	165764.2	430.37	3.53	162607.0	1358.70	1.94
30/700/100/V/L	51063.0	46.85	49699.0	50.08	2.74	49699.0	94.12	0.00
30/700/100/F/L	71063.0	46.40	67298.0	50.08	5.59	66209.0	141.47	1.64
30/700/100/V/T	48984.0	49.89	48588.0	53.53	0.82	48463.0	193.20	0.26
30/700/100/F/T	61363.0	50.74	59767.0	54.30	2.67	59500.0	146.79	0.45
30/700/400/V/L	106077.4	303.69	104270.1	356.41	1.73	103822.3	1046.73	0.43
30/700/400/F/L	160762.6	396.81	153455.0	547.94	4.76	152981.0	1170.45	0.31
30/700/400/V/T	101656.8	225.40	99032.5	446.16	2.65	99032.5	828.30	0.00
30/700/400/F/T	150867.0	290.47	145892.0	524.08	3.41	142883.2	1648.04	2.11
Average	187885.17	69.20	184844.02	107.70	2.55	183941.38	283.24	0.77

Table 3.11: Comparison of Slope Scaling variants

The presented results reveal expected behavior od Slope Scaling heuristics. Namely, regarding solution quality, as expected `SS_intens` is better than `Basic SS` as well as `iterated_SS` is better than `SS_intens`. Additionally, the average improvement achieved by `SS_intens` over `Basic SS` is 2.55%, although `SS_intens` consumed about 40 seconds more, on the average, than `Basic SS`. On the other hand, `iterated_SS` improves `SS_intens` just 0.77% on the average, but the average CPU time consumed by it is about 3 times greater than the average CPU time consumed by `SS_intens`.

Comparison with state-of-the-art heuristics

In this section we perform comparison of ILPH variants described above, with most recent heuristic approaches from the literature. The comparison is presented in Tables 3.12 and 3.13, using the following abbreviations for considered heuristics:

- `LB`: the Local Branching based heuristic of Rodriguez et al. (2010) [191],
- `IPS`: the IP search of Hewit et al. (2010) [127],
- `MIP-TS`: the MIP-tabu search of Chouman et al. (2010) [39],
- `CS`: the Capacity Scaling heuristic of Katayama et al. (2009) [141],
- `CS&LB`: the heuristic of Katayama et al. [140] that combines Capacity Scaling and Local Branching heuristics. We report results obtained by `CS&LB` for various parameter settings. So, the numbers that follow the abbreviation `CS&LB` designate the used parameter settings. In total we report results for 6 heuristics that combine Capacity Scaling and Local Branching heuristics.
- `CEA`: the Cycle-Based Evolutionary Algorithm (CEA) proposed by Paraskevopoulos et al. [179] and tested with the time limit of 20000 seconds,
- `CEA-old`: the Cycle-Based Evolutionary Algorithm (CEA) proposed by Paraskevopoulos et al. [178] and tested with the time limit of 3600 seconds,

- SACG: the heuristic of Yaghini et al. [226] that combines simulated annealing and column generation tested with the time limit of 600 seconds (SACG600) and 1800 seconds (SACG18000).

Additionally, in columns BK and Best ILPH, we report the best solution values found by heuristics from the literature and ILPH heuristics proposed here, respectively. Finally, in the column CPLEX, we report solution values offered by CPLEX 12.6 mip solver which were executed with the time limit of 3600 seconds on each test instance.

Problem	Lower Bound	Optimal	LB	IPS	MIP-TS	CS	CS&LB 5-300	CS&LB 5-900	CS&LB 10-300	CS&LB 10-900	CS&LB 15-300	CS&LB 15-900	CS&LB best	CEA	CEA-old
100/400/010/F/L		23949	24690	23949	24161	24459	24022	24022	24022	24022	23949	23949	23949	23949	23949
100/400/010/F/T	62516		67357	65885	67233	72169	66912	66912	65582	64607	65195	65112	64607	65563	66240
100/400/010/V/L		28423	28423	28423	28423	28423	28423	28423	28423	28423	28423	28423	28423	28423	28426
100/400/030/F/L		49018	49872	49694	49682	51956	50361	50361	49018	49018	49018	49018	49018	49466	49577
100/400/030/F/T	131117		141633	141365	144349	144314	141602	140206	137488	139589	140588	136446	136446	139535	139661
100/400/030/V/T		384802	384809	384836	384940	384880	384802	384802	384802	384802	384802	384802	384802	384999	385163
20/230/040/V/L		423848	423848	424385	423848	423848	423848	423848	423848	423848	423848	423848	423848	423848	423848
20/230/040/V/T		371475	371475	371779	371475	371906	371475	371475	371475	371475	371475	371475	371475	371475	371475
20/230/040/F/T		643036	643036	643187	643538	643666	643187	643187	643036	643036	643036	643036	643036	643187	643187
20/230/200/V/L		94213	95295	95097	94218	94213	94213	94213	94213	94213	94213	94213	94213	94468	94468
20/230/200/F/L		137642	143446	141253	138491	137851	137851	137642	137851	137851	137851	137851	137642	138954	139002
20/230/200/V/T		97914	98039	99410	98612	97968	97968	97914	97914	97914	97968	97914	97914	98209	98209
20/230/200/F/T	135421		141128	140273	136309	136302	136302	136031	136302	136135	136302	136302	136031	137131	137131
20/300/040/V/L		429398	429398	429398	429398	429398	429398	429398	429398	429398	429398	429398	429398	429398	429398
20/300/040/F/L		586077	586077	586077	588464	587800	586077	586077	586077	586077	586077	586077	586077	586077	586077
20/300/040/V/T		464509	464509	464509	464509	464569	464509	464509	464509	464509	464509	464509	464509	464509	464509
20/300/040/F/T		604198	604198	604198	604198	604198	604198	604198	604198	604198	604198	604198	604198	604198	604198
20/300/200/V/L	74593		76375	75319	75045	74913	74913	74830	74913	74913	74913	74913	74830	75279	75288
20/300/200/F/L	113538		119143	117543	116259	115876	115876	115751	115876	115751	115876	115876	115751	116801	117320
20/300/200/V/T		74991	76168	76198	74995	74991	74991	74991	74991	74991	74991	74991	74991	75444	75607
20/300/200/F/T	106242		109808	110344	109164	107467	107467	107467	107467	107467	107467	107467	107467	107546	108459
30/520/100/V/L		53958	54026	54113	54008	54012	53958	53958	53958	53958	53958	53958	53958	54099	54109
30/520/100/F/L	93065		96255	94388	93967	94743	94043	94043	94377	94102	94407	94405	94043	95142	95302
30/520/100/V/T		52046	52129	52174	52156	52270	52046	52046	52046	52046	52046	52046	52046	52182	52284
30/520/100/F/T	96051		101102	98883	97490	98867	98283	97789	98283	97623	98492	97377	97377	97856	98525
30/520/400/V/L	112626		114367	114042	112927	112846	112846	112786	112846	112846	112846	112846	112786	113193	113694
30/520/400/F/L	147252		157726	154218	149920	149446	149446	149446	149446	149446	149446	149446	149446	151145	151688
30/520/400/V/T		114640	115240	114922	114664	114641	114641	114641	114641	114641	114641	114641	114641	115697	116322
30/520/400/F/T	150510		168561	154606	152929	152745	152745	152745	152745	152745	152745	152745	152745	154425	154425
30/700/100/V/L		47603	47603	47612	47603	47614	47603	47603	47603	47603	47603	47603	47603	47603	47619
30/700/100/F/L	59481		60272	60700	60184	60192	59995	59995	60067	59995	60124	60067	59995	60538	60596
30/700/100/V/T		45872	45905	46046	45880	46169	45925	45925	45925	45925	45925	45875	45875	46082	46084
30/700/100/F/T	54717		55104	55609	54926	55339	54904	54904	54919	54904	55138	54951	54904	55135	55271
30/700/400/V/L	97117		103787	98718	97982	97960	97960	97960	97960	97960	97960	97960	97960	98729	99222
30/700/400/F/L	131233		169760	152576	135109	135100	135100	135100	135100	135100	135100	135100	135100	137112	137112
30/700/400/V/T	94443		96680	96168	95781	95306	95306	95306	95306	95306	95306	95306	95306	96130	96388
30/700/400/F/T	128027		144926	131629	130856	130146	130146	130146	130146	130146	130146	130146	130146	132425	133245
Average:			180058.6	178365.6	177397.1	177528.7	177117.4	177044.6	176939.8	176934.7	177026.5	176872.7	176825.8	177458.2	177650.8
Number of optimal solutions:			9	6	7	6	13	15	16	16	16	17	18	9	7

Table 3.12: Comparison with the state-of-the-art heuristics-PartI.

Problem	Lower Bound	Optimal	SACG 600	SACG 18000	ILPH_0_0	ILPH_0_01	ILPH_SS_0_0	ILPH_SS_0_0_imp	ILPH_SSimp	ILPH_CPLEX	ILPH_SS	BK	Best ILPH	CPLEX
100/400/010/F/L		23949	23949	23949	23949	24299	24299	24299	23949	23949	24912	23949	23949	23949
100/400/010/F/T	62516		65172	65172	64245	67341	65550	65550	65247	65247	69218	64607	64245	63753
100/400/010/V/L		28423	28423	28423	28423	28423	28423	28423	28423	28423	28423	28423	28423	28423
100/400/030/F/L		49018	49641	49250	49018	49568	49279	49279	49018	49018	52564	49018	49018	49115
100/400/030/F/T	131117		141538	141014	137803	139092	139283	138470	139177	138784	140798	136446	137803	139016
100/400/030/V/T		384802	384802	384802	384802	384802	384802	384802	384802	384802	384802	384802	384802	384802
20/230/040/V/L		423848	423848	423848	423848	423848	423848	423848	423848	423848	424612	423848	423848	423848
20/230/040/V/T		371475	371475	371475	371475	371475	371475	371475	371475	371475	371907	371475	371475	371475
20/230/040/F/T		643036	643036	643036	643036	643036	643036	643036	643036	643036	643740	643036	643036	643036
20/230/200/V/L		94213	95295	94283	94218	94218	94363	94218	94213	94218	97068	94213	94213	94218
20/230/200/F/L		137642	139762	137842	138555	138270	137854	138270	138169	138348	145564	137642	137854	138491
20/230/200/V/T		97914	97984	97914	97914	97914	97914	97914	97914	97914	100378	97914	97914	97914
20/230/200/F/T	135421		140763	137072	137128	136102	136812	136102	136513	136102	141064	136031	136102	136338
20/300/040/V/L		429398	429398	429398	429398	429398	429398	429398	429398	429398	429398	429398	429398	429398
20/300/040/F/L		586077	586077	586077	586077	586077	586077	586077	586077	586077	587705	586077	586077	586077
20/300/040/V/T		464509	464627	464627	464509	464509	464509	464509	464509	464509	464550	464509	464509	464509
20/300/040/F/T		604198	604201	604201	604198	604198	604198	604198	604198	604198	604821	604198	604198	604198
20/300/200/V/L	74593		76003	74902	74996	74976	74987	75055	74971	75003	78232	74830	74971	74929
20/300/200/F/L	113538		117942	116431	116620	116712	116352	116712	116375	116759	120063	115751	116352	116264
20/300/200/V/T		74991	75419	74991	74991	74991	74991	74991	74991	74991	77394	74991	74991	74991
20/300/200/F/T	106242		109710	108638	107703	107492	107192	107324	107298	107102	109425	107467	107102	107709
30/520/100/V/L		53958	53983	53983	53958	53958	53958	53958	53958	53958	54657	53958	53958	53958
30/520/100/F/L	93065		94307	94066	94043	94033	93967	94167	94066	93991	98152	93967	93967	94043
30/520/100/V/T		52046	52390	52247	52046	52046	52046	52046	52046	52046	52504	52046	52046	52046
30/520/100/F/T	96051		100184	98543	97413	97630	97167	97630	97404	97711	101383	97377	97167	97614
30/520/400/V/L	112626		114201	113720	113138	113006	113033	113006	112974	112957	114784	112786	112957	112922
30/520/400/F/L	147252		155837	151009	150646	151005	151061	151005	149945	151134	152409	149446	149945	153028
30/520/400/V/T		114640	115614	115581	114855	114945	114833	114863	114798	114757	116036	114641	114757	114842
30/520/400/F/T	150510		159976	147369	155373	155395	155226	154637	153856	154859	155656	*147369*	153856	174937
30/700/100/V/L		47603	47998	47603	47603	47603	47603	47603	47603	47603	48337	47603	47603	47603
30/700/100/F/L	59481		60581	60391	60023	60036	60017	60036	60049	60011	63067	59995	60011	60124
30/700/100/V/T		45872	46147	45956	45908	45872	45872	45872	45908	45908	46322	45875	45872	45872
30/700/100/F/T	54717		55012	54975	54904	54904	54904	54904	54904	54904	55788	54904	54904	55055
30/700/400/V/L	97117		101726	99316	98518	98746	98430	98507	98385	98534	100345	97960	98385	113315
30/700/400/F/L	131233		160522	133976	141891	141516	139842	141889	139663	141170	139663	133976	139663	169047
30/700/400/V/T	94443		97046	95538	96181	95968	95746	95733	95773	95863	96708	95306	95733	97857
30/700/400/F/T	128027		141491	131473	131989	130589	130871	131284	131141	131814	132709	130146	130589	149408
Average:			179083.2	177110.6	177335.0	177405.2	177276.2	177326.8	177191.2	177308.7	179058.3	176648.1	177072.8	179570.9
Number of optimal solutions:			8	11	16	15	15	15	17	16	3	18	18	15

Table 3.13: Comparison with the state-of-the-art heuristics-PartII.

From the reported results the following conclusions may be drawn:

- The proposed ILPH based heuristics are highly competitive with the existing heuristic approaches from the literature, although some of them have been executed with time limit greater than 3600 seconds (e.g., `CS&LB, CEA, SACG18000`). Additionally, it should be emphasized that `SACG18000` on the test instance 30/520/400/F/T reported the upper bound value that is less than the corresponding lower bound. Thus, the validity of all other values reported by `SACG` is also questionable.

- The proposed ILPH variants succeed to improve the previous upper bounds for the following test instances: 100/400/010/F/T, 20/300/200/F/T, 30/520/100/F/T, 30/700/100/V/T. Two of four new best known upper bounds are due to `ILPH_SS_0_0` (instances 30/520/100/F/T, 30/700/100/V/T), while `ILPH_0_0` and `ILPH_CPLEX` improved upper bounds for instances 100/400/010/F/T and 20/300/200/F/T, respectively.

- Regarding the number of reached optimal solutions for instances solved to optimality, we infer that ILPH heuristics, all together, are able to reproduce the same number of optimal solutions as all existing heuristics are able to do together (i.e., 18 out of 20 optimal solutions). Furthermore, each ILPH based heuristic, except `ILPH_SS`, as well as each `CS&LB` heuristic, except `CS&LB 5-300`, is able to reproduce at least 15 optimal solutions, i.e., significantly more than any other heuristic.

- Regarding the average solution value offered by tested ILPH based heuristics we conclude that `ILPH_CPLEX` and `ILPH_SS_0_0` are two best approaches, while `ILPH_SS` is the worst one. Such ranking of `ILPH_SS` was expected since it uses simple Slope Scaling heuristic to solve reduced problem arising in an iteration of ILPH unlike any other ILPH heuristic that uses more sophisticated tool for that purposes.

3.4 Concluding remarks

In this chapter we propose two new heuristics for finding initial feasible solutions of mixed integer programs (MIPs). The proposed heuristics, called

variable neighborhood diving (*VN diving*) and *single neighborhood diving* (*SN diving*), perform systematic hard variable fixing (i.e., diving) in order to generate smaller subproblems whose feasible solution (if one exists) is also feasible for the original problem. In VN diving, this fixation is performed according to the rules of variable neighborhood decomposition search (VNDS). This means that a number of subproblems (neighborhoods) generated in a VNDS manner are explored in each iteration. Also, pseudo-cuts are added during the search process in order to prevent exploration of already visited search space areas. However, a feasible solution is usually obtained in the first iteration. In SN diving, only one neighborhood is explored in each iteration. However, we introduce a new mechanism to avoid the already visited solutions. It consists of memorizing a set of constraints in a new MIP problem, which is then solved instead of the original problem in order to obtain the new reference solution. Our experiments show that this mechanism generally provides much better diversification than the addition of pseudo-cuts alone. Moreover, we have proved that the SN diving algorithm converges to a feasible solution, if one exists, or proves the infeasibility in a finite number of iterations. Both methods use the generic CPLEX MIP solver as a black-box for tackling the subproblems generated during the search.

The proposed heuristics are tested on two established sets of benchmark instances, proven to be difficult: the first set contains 83 0-1 MIP instances, and the second contains 34 general MIP instances. We compare our heuristics with the IBM ILOG CPLEX 12.4 MIP solver, the two variants of the feasibility pump (FP) heuristic (standard FP and objective FP), and the variable neighborhood pump (VNP) heuristic. According to an extensive experimental analysis, both VN and SN diving clearly outperform the CPLEX MIP solver and the two FP heuristics regarding the solution quality, within a similar or shorter computational time. Additionally, on the instances from General MIP benchmark, SN diving performs better than any other tested method, regarding not only solution quality but also the time needed to find a feasible solution. The reported results reported are also competitive with those obtained by the recent variable neighborhood pump heuristic. Besides improving the basic variable neighborhood pump, our future work may consist of designing a multi-objective VNS heuristic, which would tackle

both infeasibility and original objective quality during the search process.

Additionally, in this chapter we studied Multicommodity Fixed-Charge Network Design (MCND) problem. We propose several Iterative linear programming-based heuristics for solving this NP-hard problem. Additionally, we propose how to adapt well-known Slope Scaling heuristic for MCND in order to tackle reduced problems of MCND obtained by fixing some binary variables. Moreover, we show that ideas of a convergent algorithm based on the LP-relaxation and pseudo-cuts may be combined by those of a Slope Scaling heuristic during the search for an optimal (or near-optimal) solution. Also, it is worth mentioning that this is the first time that some convergent algorithm based on the LP-relaxation and pseudo-cuts algorithm has been considered as a tool for solving some general 0-1 MIP problem.

The proposed heuristics have been tested on the benchmark instances from the literature. The quality of solutions obtained by each of them has been disclosed comparing them with the current state-of-the-art heuristics. The computational results show that the proposed approaches are competitive with current state-of-the-art heuristics. In particular, proposed approaches are able to reproduce 18 optimum solutions for 20 instances previously solved by exact algorithms. In addition, the proposed algorithms offered four new best known heuristic solutions.

Chapter 4

Scatter Search and Star Paths with Directional Rounding for 0–1 Mixed Integer Programs

4.1 Introduction

A 0–1 mixed integer program (MIP) may be written in the following form:

$$\begin{array}{ll} \textit{minimize} & z = cx \\ \textit{s.t.} & Ax = b \\ & 0 \leq x_j \leq U_j,\ j \in N = \{1, \ldots, n\} \\ & x_j \in \{0, 1\},\ j \in \mathcal{I} \subseteq N \end{array} \tag{4.1}$$

where A is a constant matrix, b is a constant vector, the set N denotes the index set of variables, while the set $\mathcal{I}$ contains indices of binary variables. Each variable x_j has an upper bound denoted by U_j (which equals 1 if x_j is binary, and otherwise may be infinite). It is assumed that all continuous variables can be represented (either directly or by transformation) as slack variables, i.e., the associated columns of the (possibly transformed) matrix A constitute an identity matrix. Hence, if the values of binary variables are known, the continuous variables receive their values automatically. The relaxation of MIP obtained by excluding integrality constraints will be denoted by LP. A feasible solution of $MIP(LP)$ will be called $MIP(LP)$ feasible.

The Scatter Search evolutionary metaheuristic combines decision rules and problem constraints, and it has its origins in surrogate constraint strategies. Scatter Search, unlike Genetic Algorithms, operates on a small set of solutions and makes only limited use of randomization as a proxy for diversification when searching for a globally optimal solution. Since its introduction by Fred

Glover in 1965 [87, 90] as a heuristic for integer programming, Scatter Search has been successfully applied to a wide range of combinatorial optimization problems. The basic Scatter Search design, which can be implemented in varying degrees of sophistication, can be expressed in terms of a five method template ([53, 95, 96, 102, 105, 148, 149, 161, 190]):

i) A Diversification Generation Method to generate a collection of diverse trial solutions within the search space.

ii) An Improvement Method to transform a trial solution into one or more enhanced trial solutions.

iii) A Reference Set Update Method to build and maintain a reference set consisting of the β best solutions found, where the value of β is typically small, e.g., no more than 20. Solutions gain membership to the reference set according to their quality or their diversity.

iv) A Subset Generation Method to operate on the reference set, to produce several subsets of its solutions as a basis for creating combined solutions.

v) A Solution Combination Method to transform a given subset of solutions produced by the Subset Generation Method into one or more combined solution vectors.

Scatter search and its Path Relinking generalization have been been successfully applied in a wide range of discrete and nonlinear optimization settings including neural networks, routing problems, graph drawing, scheduling, linear ordering, assignment, p–Median, knapsack, coloring problems, clustering / selection and software testing (see for example [190]).

The introduction of Star Paths with directional rounding for 0–1 Mixed Integer Program as a supporting strategy for Scatter Search in [93] established basic properties of directional rounding and provided efficient methods for exploiting them. The most important of these properties is the existence of a plane (which can be required to be a valid cutting plane for MIP) which contains a point that can be directionally rounded to yield an optimal

solution and which, in addition, contains a convex subregion all of whose points directionally round to give this optimal solution. Several alternatives are given for creating such a plane as well as a procedure to explore it using principles of Scatter Search. This work also shows that the set of all 0–1 solutions obtained by directionally rounding points of a given line (the so–called Star Path) contains a finite number of different 0–1 solutions and provides a method to generate these solutions efficiently. Glover and Laguna [101] elaborate these ideas and extend them to General Mixed Integer Programs by means of a more general definition of directional rounding.

Building on these ideas, Glover et al. [106] propose a procedure that combines Scatter Search and the Star Path generation method as a basis for finding a diverse set of feasible solutions for 0–1 MIP, proposing a 3–phase algorithm which works as follows. The first step generates a diverse set of 0–1 solutions using a dichotomy generator. After that each solution generated in a previous phase is used to produce two center points on an LP polyhedron which are further combined to produce sub-centers. All centers and sub–centers are combined in the last phase to produce Star Paths. As output the algorithm gives the set of all 0–1 feasible solutions encountered during its execution, and constitutes the required diverse set. The computational efficiency of the approach was demonstrated by tests carried out on some instances from MIPLIB.

In this chapter, we prove all theorems stated in [93] and also we give some corrections and extensions of that work. We extend these previous contributions to provide new results that expand the fundamental properties established for Star Paths and introduce new MIP methods that exploit these results. Additionally, we describe connection between continuous and discrete solution space and how we can project from continuous to discrete space (i.e., directional rounding of a cone).

The main contribution of our work is proposing Convergent algorithms of Scatter Search and Star Paths with Directional Rounding for 0-1 MIP and proving their finite convergence. Note that this is the first time that the finite convergence of a Scatter Search algorithm is proven. Moreover, we propose several implementations of the Convergent Scatter Search algorithm and

illustrate their running on small examples. Additionally, we propose several One-Pass (non-iterated) heuristics based on Scatter Search and directional rounding. The versions of the methods tested are "first stage" implementations to establish the power of the methods in a simplified form. The aim of this part is to demonstrate the efficacy of these first stage methods, which makes them attractive for use in situations where very high quality solutions are sought with an efficient investment of computational effort.

The rest of the chapter is organized as follows. Section 4.2 begins by proving key properties of directional rounding and provides an efficient method for directional rounding of a line, hence for generating Star Paths. In Section 4.3 we state and prove several theorems which enable us to organize search for an optimal solution of 0–1 MIP problems using Scatter Search. In Section 4.4, we propose Convergent Scatter Search Algorithms and we provide the proof of their finite convergence. Additionally, we give examples to illustrate the execution of those Convergent Scatter Search Algorithms based on directional rounding. In Section 4.5, we propose a framework for One-Pass Scatter Search heuristics and perform an empirical study in order to find the best ingredients for a One-Pass heuristic. Section 4.6 describes a heuristic derived from Convergent Scatter Search. Finally, in Section 4.7 we present comparisons of solutions obtained by our heuristics with the best known solutions for the problems tested, and present our conclusions in Section 4.8.

4.2 Generating Star Paths with directional rounding

4.2.1 Basic Notation

For two arbitrary points x' and x'', we identify:
the ray from x' through x'' by

$$Ray(x', x'') = \{x' + \lambda(x'' - x') : \lambda \geq 0\};$$

the line joining x' and x'' by

$$Line(x', x'') = \{x' + \lambda(x'' - x') : \lambda \in \mathbb{R}\};$$

and the segment with extremities x' and x'' by

$$[x', x''] = \{x' + \lambda(x'' - x') : 0 \leq \lambda \leq 1\}.$$

Let $X(R)$ denote a chosen set of reference points, indexed by the set R i.e., $X(R) = \{x(r) : r \in R\}$. Since, the points of $X(R)$ are linearly independent in the usual case to be considered, we define the hyperplane consisting of all normalized linear combinations of these points by:

$$Plane(X(R)) = \{\sum_{r \in R} \lambda_r x(r) : \sum_{r \in R} \lambda_r = 1\}. \tag{4.2}$$

Furthermore, we identify the associated half space as:

$$Half\_space(X(R)) = \{\sum_{r \in R} \lambda_r x(r) : \sum_{r \in R} \lambda_r \geq 1\}. \tag{4.3}$$

If we choose a point x^* which does not belong to the $Plane(X(R))$ and which therefore constitutes a set of affinely independent points together with points from $X(R)$, we will be able to define the polyhedral (half) cone spanned by the rays from x^* through the points of $X(R)$:

$$Cone(x^*, X(R)) = \{x^* + \sum_{r \in R} \lambda_r (x(r) - x^*) : \lambda_r \geq 0, r \in R\}. \tag{4.4}$$

Finally, the intersection of $Cone(x^*, X(R))$ with $Half\_space(X(R))$, i.e., the face of the truncated cone that excludes the point x^* is defined as:

$$Face(X(R)) = \{\sum_{r \in R} \lambda_r x(r) : \sum_{r \in R} \lambda_r = 1,\ \lambda_r \geq 0, r \in R\}. \tag{4.5}$$

The Hamming distance between two solutions, $x' \in \{0, 1\}^n$ and $x \in [0, 1]^n$

can be represented as:

$$d(x', x) = \sum_{j=1}^{n} |x'_j - x_j| = x(e - x') + x'(e - x), \tag{4.6}$$

where $e = (1, 1, \ldots, 1)$ is the vector of all 1's with appropriate dimension.

Rounding and complementing operators are useful in mathematical programming with 0–1 variables. We will show that the directional rounding introduced by Glover (1993) provides an extension of the rounding and complementing operators.

4.2.2 Definition and properties of directional rounding

Directional rounding is a mapping δ from the continuous space $[0, 1]^n \times [0, 1]^n$ to the discrete space $\{0, 1\}^n$ by the following rules. The directional rounding $\delta(x^*, x')$, from a base point $x^* \in [0, 1]^n$ to an arbitrary focal point x', is the point in $\{0, 1\}^n$ given by

$$\delta(x^*, x') = (\delta(x^*_j, x'_j) : j \in N)$$

where $\delta(x^*_j, x'_j)$ is defined as

$$\delta(x^*_j, x'_j) = \begin{cases} 0 & \text{if } x'_j < x^*_j \\ 1 & \text{if } x'_j > x^*_j \\ x^*_j & \text{if } x'_j = x^*_j \in \{0, 1\} \\ 0 \text{ or } 1 & \text{if } x'_j = x^*_j \notin \{0, 1\} \end{cases} \tag{4.7}$$

It may be noted that directional rounding includes nearest neighbor rounding as a special instance, i.e., $near(x'_j) = \delta(0.5, x'_j)$. The directional rounding in the last case $x'_j = x^*_j \notin \{0, 1\}$ can be performed in several ways. Some of them are:

- use simple rounding, i.e., $\delta(x^*_j, x^*_j) = near(x^*_j)$ breaking the tie arbitrarily for $x^*_j = 0.5$,
- choose randomly 0 or 1,

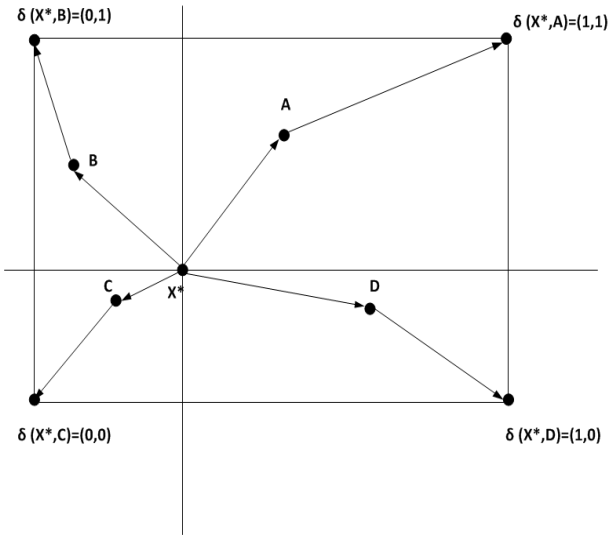

Figure 4.1: Example of directional rounding in two-dimensional case

- make the choice according to values of x_j^* and x_j' in the following way. If $x_j' \leq 1/3$ or $2/3 \leq x_j' \leq 1$ perform simple rounding, otherwise choose randomly 0 or 1,
- choose a segment $[y', y'']$ which contains x^* and after that set $\delta(x_j^*, x_j^*) = 1$ if $y_j' < y_j''$ while otherwise set $\delta(x_j^*, x_j^*) = 0$.

Figure 4.1 illustrates directional rounding for different focal points labeled A, B, C, and D.

We define the directional rounding $\delta(x^*, X)$ from the point x^* to the set X to be the set of points in $\{0,1\}^n$ given by

$$\delta(x^*, X) = \{\delta(x^*, x') : x' \in X\}.$$

The motivation for directional rounding comes from the fact that if x^* is an extreme point of the feasible set of LP-relaxation, then every feasible 0-1 solution can be obtained by directional rounding relative to focal points that

lie within the LP cone defined by non-negative values for the current nonbasic variables (as proved in section 4.3).

As stated in the next two lemmas, for any vertex of the unit hypercube $x' \in \{0,1\}^n$, we have $\delta(x^*, x') = x'$, while directional rounding of any point x' relatively to the base point $x^* \in \{0,1\}^n$ yields a point deduced from x^* by complementing only those coordinates that have different values in x^* and x'.

Lemma 4.2.1 *For any base point $x^* \in [0,1]^n$ and focal point $x' \in \{0,1\}^n$*

$$\delta(x^*, x') = x'.$$

Proof. From the definition of directional rounding we have $\delta(x_j^*, 1) = 1$ and $\delta(x_j^*, 0) = 0$, hence $\delta(x_j^*, x_j') = x_j'$ for $x_j' \in \{0,1\}$. So, $\delta(x^*, x') = x'$ for each point $x' \in \{0,1\}^n$. □

Lemma 4.2.2 *For any base point $x^* \in \{0,1\}^n$ and focal point $x' \in [0,1]^n$*

$$\delta(x_j^*, x_j') = \begin{cases} 1 - x_j^* \text{ if } x_j' \neq x_j^* \\ x_j^* \text{ otherwise} \end{cases} \tag{4.8}$$

Proof. If $x_j' < x_j^*$ then x_j^* must equal 1, so $\delta(x_j^*, x_j') = 0$, which is equal to $1 - x_j^*$. Similarly, if $x_j' > x_j^*$ then x_j^* must equal 0 and therefore $\delta(x_j^*, x_j') = 1 - x_j^*$. Finally, in the case $x_j' = x_j^*$ the definition of directional rounding for the case $x_j' = x_j^* \in \{0,1\}^n$ gives $\delta(x_j^*, x_j') = x_j^*$. □

Corollary 4.2.3 *For any base point $x^* \in \{0,1\}^n$ and focal point $x' \in]0,1[^n$*

$$\delta(x^*, x') = e - x^*.$$

The next lemma says that every point on the ray from x^* through x' (excluding the point x^* itself) gives the same directional rounding as $\delta(x^*, x')$.

Lemma 4.2.4 *For any base point $x^* \in [0,1]^n$ and a focal point $x' \neq x^*$*

$$\delta(x^*, x') = \delta(x^*, x'')$$

for all $x'' \in Ray(x^*, x')$ *such that* $x'' \neq x^*$.

Proof. Since $x'' \in Ray(x^*, x')$ there exists $\lambda \geq 0$ such that $x'' = \lambda x' + (1-\lambda)x^*$. Therefore for each component x''_j, $j \in N$, we have the three cases:

$$\begin{cases} i) x'_j < x^*_j \ \Rightarrow x''_j < x^*_j, \\ ii) x'_j > x^*_j \ \Rightarrow x''_j > x^*_j, \\ iii) x'_j = x^*_j \ \Rightarrow x''_j = x^*_j, \end{cases} \tag{4.9}$$

These observations lead to the conclusion that $\delta(x^*_j, x'_j) = \delta(x^*_j, x''_j)$ for each $j \in N$ which further implies $\delta(x^*, x') = \delta(x^*, x'')$. □

4.2.3 Efficient procedure for generating a Star Path

A Star Path $L(x^*, x', x'')$ is defined as a set of 0-1 vectors obtained by directional rounding using points which belong to the line connecting x' and x'' as focal points and x^* as a base point. More precisely,

$$L(x^*, x', x'') = \delta(x^*, Line(x', x'')).$$

In this section, we provide several properties which lead to an efficient procedure to compute $\delta(x^*, S)$ where S is a subset of $Line(x', x'')$.

Lemma 4.2.5 *Given* x' *and* x''*, as focal points and* x^* *as a base point, define* $\lambda^*_j = \frac{x^*_j - x'_j}{x''_j - x'_j}$ *for* $j \in N^{\neq} = \{j \in N : x''_j \neq x'_j\}$*. For every real value of* λ*, the elements of the vector* $\delta(x^*, x' + \lambda(x'' - x'))$ *in* $L(x^*, x', x'')$ *are given by*

$$\delta(x^*_j, x'_j + \lambda(x''_j - x'_j)) = \begin{cases} \delta(x^*_j, x'_j) & \text{if } x''_j = x'_j \\ 1 & \text{if } (\lambda < \lambda^*_j \text{ and } x''_j < x'_j) \text{ or } (\lambda > \lambda^*_j \text{ and } x''_j > x'_j) \\ 0 & \text{if } (\lambda < \lambda^*_j \text{ and } x''_j > x'_j) \text{ or } (\lambda > \lambda^*_j \text{ and } x''_j < x'_j) \\ \delta(x^*_j, x^*_j) & \text{if } (x''_j \neq x'_j) \text{ and } (\lambda = \lambda^*_j) \end{cases} \tag{4.10}$$

Proof. Let $x_j = x'_j + \lambda(x''_j - x'_j)$. The case where $x'_j = x''_j$ is obvious. For the case where $x''_j > x'_j$, the condition $\lambda^*_j < \lambda$ implies

$$\lambda^*_j = \frac{x^*_j - x'_j}{x''_j - x'_j} < \lambda = \frac{x_j - x'_j}{x''_j - x'_j}.$$

Hence $x^*_j < x_j$ which means that $\delta(x^*_j, x_j) = 1$, and the condition $\lambda^*_j > \lambda$ implies $x^*_j > x_j$ which means that $\delta(x^*_j, x_j) = 0$. Similarly, in the case $x''_j < x'_j$, we have $\delta(x^*_j, x_j) = 1$ if $\lambda^*_j > \lambda$ and $\delta(x^*_j, x_j) = 0$ if $\lambda^*_j < \lambda$. In the last case where $\lambda = \lambda^*_j$ we have $x_j = x^*_j$. □

The definition of the element $\delta(x^*_j, x'_j + \lambda(x''_j - x'_j))$ needs a specific "tie breaking" rule to handle the case $\lambda = \lambda^*_j$ (which corresponds to $\delta(x^*_j, x'_j + \lambda(x''_j - x'_j) = \delta(x^*_j, x^*_j)$), where the original definition of directional rounding requires such a rule to choose between a value of 0 or 1, in our implementation, we use simple rounding. The proof of the preceding Lemma 4.2.5 does not depend on the restriction on λ.

Let $(\pi(1), \pi(2), \ldots, \pi(t))$ be a permutation of the indexes of $N^{\neq} = \{j \in N : x''_j \neq x'_j\}$ so that $\lambda^*_{\pi(1)} \leq \lambda^*_{\pi(2)} \leq \cdots \leq \lambda^*_{\pi(t)}$, where $t = |N^{\neq}|$. We will assume that the $\lambda^*_{\pi(h)}$ values are all distinct so that $\lambda^*_{\pi(h)} < \lambda^*_{\pi(h+1)}$ for all $h < t$.

Theorem 4.2.1 *Given x' and x'', as focal points and x^* as a base point, let $\lambda^*_j = \frac{x^*_j - x'_j}{x''_j - x'_j}$ for $j \in N^{\neq}$ and define $l(\lambda) = \delta(x^*, x' + \lambda(x'' - x'))$. Then for each $h = 1, 2, \ldots, |N^{\neq}| - 1$ and for every λ and λ' such that $\lambda, \lambda' \in]\lambda^*_{\pi(h)}, \lambda^*_{\pi(h+1)}[$, we have*

$$l(\lambda) = l(\lambda').$$

Moreover, $d(l(\lambda^*_{\pi(h+1)}), l(\lambda^*_{\pi(h)})) \leq 2$, ($d(\cdot, \cdot)$ represents Hamming distance operator), and more precisely

$$l_j(\lambda^*_{\pi(h+1)}) = l_j(\lambda^*_{\pi(h)}) \, for \, j \in N - \{\pi(h), \pi(h+1)\},$$

$$l_{\pi(h)}(\lambda^*_{\pi(h+1)}) = \begin{cases} 0 & \text{if } x''_{\pi(h)} < x'_{\pi(h)} \\ 1 & \text{if } x''_{\pi(h)} > x'_{\pi(h)} \end{cases} \quad \text{and } l_{\pi(h+1)}(\lambda^*_{\pi(h+1)}) = \delta(x^*_{\pi(h+1)}, x^*_{\pi(h+1)}). \tag{4.11}$$

Proof. According to Lemma 4.2.5, the value $l_j(\lambda)$ for $j \in N^= = \{j \in N : x''_j = x'_j\}$ does not depend on λ and is the same for all points from the line connecting points x' and x''. Hence $l_j(\lambda) = l_j(\lambda')$ for $j \in N^=$. From the assumption that $\lambda^*_{\pi(h)} < \lambda^*_{\pi(h+1)}$ for all $h < |N^{\neq}|$, we can partition the set $N^{\neq}$ into two sets $N' = \{j \in N^{\neq} : \lambda^*_j \leq \lambda^*_{\pi(h)}\}$ and $N'' = \{j \in N^{\neq} : \lambda^*_j \geq \lambda^*_{\pi(h+1)}\}$. For $j \in N'$ implies $\lambda^*_j < \lambda$, and $\lambda^*_j < \lambda'$ hence by Lemma 4.2.5, $l_j(\lambda) = l_j(\lambda')$. Similarly, $j \in N''$ implies $\lambda^*_j \geq \lambda^*_{\pi(h+1)} > \lambda > \lambda^*_{\pi(h)}$, and $\lambda' > \lambda^*_{\pi(h)}$ and hence holds $l_j(\lambda) = l_j(\lambda')$. So, $l_j(\lambda) = l_j(\lambda')$ for each $j \in N^{\neq}$. Similarly for $j \in N^{\neq} - \{\pi(h), \pi(h+1)\}$ it follows that

$$l_j(\lambda^*_{\pi(h+1)}) = l_j(\lambda^*_{\pi(h)}).$$

Indeed, for each j such that $\lambda^*_j < \lambda^*_{\pi(h)} < \lambda^*_{\pi(h+1)}$ we have $l_j(\lambda^*_{\pi(h+1)}) = l_j(\lambda^*_{\pi(h)}) = 1$ if $x''_j > x'_j$ and $l_j(\lambda^*_{\pi(h+1)}) = l_j(\lambda^*_{\pi(h)}) = 0$ otherwise. Likewise for each j such that $\lambda^*_j > \lambda^*_{\pi(h+1)} > \lambda^*_{\pi(h)}$ we obtain $l_j(\lambda^*_{\pi(h+1)}) = l_j(\lambda^*_{\pi(h)})$. Finally, the values of $l_j(\lambda^*_{\pi(h+1)})$ for $j \in \{\pi(h), \pi(h+1)\}$, are deduced from the definition of δ. □

We show in the following example that the distance $d(l(\lambda^*_{\pi(h+1)}), l(\lambda^*_{\pi(h)}))$ can be equal to 2.

Example 1 *Consider the following simple example in two-dimension where the base point $x^* = (0, 1)$, the focal points are given by $x' = (0.1, 0.8)$ and $x'' = (\frac{2}{30}, 0.6)$, with the corresponding values $\lambda^*_1 = 3$ and $\lambda^*_2 = -1$. It is easy to see that $l(\lambda^*_1) = (0, 0)$ and $l(\lambda^*_2) = (1, 1)$. So, the Hamming distance between the two vectors $l(\lambda^*_1)$ and $l(\lambda^*_2)$ is equal to 2.*

Example 2 *In contrast to the preceding example, suppose the base point is $x^* = (0.3, 0.4)$ and the focal points are given by $x' = (0.2, 0.3)$ and $x'' = (0.4, 0.7)$. Then, the values of λ^*_1 and λ^*_2 are $\lambda^*_1 = 0.5$, $\lambda^*_2 = 0.25$.*

Hence, it follows that $l(\lambda_1^*) = (0,1)$ *and* $l(\lambda_2^*) = (1,1)$*. So, in this case the Hamming distance between the two vectors* $l(\lambda_1^*)$ *and* $l(\lambda_2^*)$ *is equal to 1.*

Lemma 4.2.6 *The assumption* $0 < \lambda_j^* < 1$*, is equivalent to the condition that the element* x_j^* *is on the open segment between the elements* x_j' *and* x_j''*, i.e.,* $min\{x_j', x_j''\} < x_j^* < max\{x_j', x_j''\}$*.*

Proof. Since λ_j^* is defined only if $x_j' \neq x_j''$, we consider the following two cases.
Case 1: $(x_j'' > x_j') \Leftrightarrow (x_j'' - x_j') > 0$.
Using the definition of λ_j^* and assumption $0 < \lambda_j^* < 1$ we obtain the following set of equivalences,

$$\lambda_j^* < 1 \Leftrightarrow \frac{x_j^* - x_j'}{x_j'' - x_j'} < 1 \Leftrightarrow (x_j^* - x_j') < (x_j'' - x_j') \Leftrightarrow x_j^* < x_j''$$

$$\lambda_j^* > 0 \Leftrightarrow \frac{x_j^* - x_j'}{x_j'' - x_j'} > 0 \Leftrightarrow (x_j^* - x_j') > 0 \Leftrightarrow x_j^* > x_j'$$

which lead into conclusion that

$$(x_j'' > x_j') \text{ and } (0 < \lambda_j^* < 1) \Leftrightarrow x_j' < x_j^* < x_j''.$$

Case 2: $(x_j'' < x_j') \Leftrightarrow (x_j'' - x_j') < 0$.
Using the same facts as in the previous case, we obtain the following relations

$$\lambda_j^* < 1 \Leftrightarrow \frac{x_j^* - x_j'}{x_j'' - x_j'} < 1 \Leftrightarrow (x_j^* - x_j') > (x_j'' - x_j') \Leftrightarrow x_j^* > x_j''$$

$$\lambda_j^* > 0 \Leftrightarrow \frac{x_j^* - x_j'}{x_j'' - x_j'} > 0 \Leftrightarrow (x_j^* - x_j') < 0 \Leftrightarrow x_j^* < x_j'$$

which imply

$$(x_j'' < x_j') \text{ and } (0 < \lambda_j^* < 1) \Leftrightarrow x_j'' < x_j^* < x_j'$$

Finally, the conclusions derived in these two cases imply that the condition $0 < \lambda_j^* < 1$ is equivalent to the condition that x_j^* is on the segment between the elements x_j' and x_j''. □

The next corollary shows that under stronger assumptions the distance between two consecutive points $l(\lambda^*_{\pi(h)})$ and $l(\lambda^*_{\pi(h+1)})$ is equal to 1.

Corollary 4.2.7 *Given x' and x'' as focal points and x^* as base point, let $\lambda_j^* = \frac{x_j^* - x_j'}{x_j'' - x_j'}$ for $j \in N^{\neq}$. Assume $0 < \lambda^*_{\pi(1)} < \lambda^*_{\pi(2)}, \ldots, < \lambda^*_{\pi(t)} < 1$ with $t = |N^{\neq}|$, and suppose further that ties in directional rounding $\delta(x_j^*, x_j^*)$ which arises in the case $0 < x_j^* < 1$ are broken in the following way*

$$\delta(x_j^*, x_j^*) = \begin{cases} 0 & \text{if } x_j'' < x_j' \\ 1 & \text{if } x_j'' > x_j' \end{cases} \tag{4.12}$$

Then the distance between vectors $l(\lambda^*_{\pi(h)})$ and $l(\lambda^*_{\pi(h+1)})$ satisfies

$$d(l(\lambda^*_{\pi(h)}), l(\lambda^*_{\pi(h+1)})) = 1.$$

More precisely,

$$l_j(\lambda^*_{\pi(h+1)}) = l_j(\lambda^*_{\pi(h)}) \, for \, j \in N - \{\pi(h+1)\}$$

$$l_{\pi(h+1)}(\lambda^*_{\pi(h+1)}) = 1 - l_{\pi(h+1)}(\lambda^*_{\pi(h)}).$$

Proof. The condition $0 < \lambda^*_{\pi(1)} < \lambda^*_{\pi(2)}, \ldots, < \lambda^*_{\pi(t)} < 1$ implies $0 < x^*_{\pi(i)} < 1, i = 1, \ldots t$, because as previously shown (Lemma 4.2.6) the condition $0 < \lambda_j^* < 1$ compels x_j^* to belong to the segment between x_j'' and x_j'. Hence, $l_j(\lambda)$ may be rewritten using its definition in Theorem 4.2.5 and the tie breaking rule as:

$$l_j(\lambda) = \begin{cases} \delta(x_j^*, x_j') & \text{if } x_j' = x_j'' \\ 0 & \text{if } (\lambda \geq \lambda_j^* \text{ and } x_j'' < x_j') \text{ or } (\lambda < \lambda_j^* \text{ and } x_j'' > x_j') \\ 1 & \text{if } (\lambda \geq \lambda_j^* \text{ and } x_j'' > x_j') \text{ or } (\lambda < \lambda_j^* \text{ and } x_j'' < x_j') \end{cases} \tag{4.13}$$

So, $l_{\pi(h)}(\lambda^*_{\pi(h)})$ equals 0 if $x''_{\pi(h)} < x'_{\pi(h)}$, and equals 1 if $x''_{\pi(h)} > x'_{\pi(h)}$. On the other hand, $l_{\pi(h)}(\lambda^*_{\pi(h+1)})$ equals 0 if $x''_{\pi(h)} < x'_{\pi(h)}$, and otherwise, equals 1. Therefore, $l_{\pi(h)}(\lambda^*_{\pi(h)}) = l_{\pi(h)}(\lambda^*_{\pi(h+1)})$. Further, $l_{\pi(h+1)}(\lambda^*_{\pi(h)})$ equals 0 if $x''_{\pi(h+1)} > x'_{\pi(h+1)}$, and otherwise it equals 1, while $l_{\pi(h+1)}(\lambda^*_{\pi(h+1)})$ equals 1 if $x''_{\pi(h+1)} > x'_{\pi(h+1)}$ and otherwise, it equals 0. Hence,

$$l_{\pi(h+1)}(\lambda^*_{\pi(h+1)}) = 1 - l_{\pi(h+1)}(\lambda^*_{\pi(h)}).$$

These conclusions together with Theorem 4.2.1 demonstrate that

$$l_j(\lambda^*_{\pi(h+1)}) = l_j(\lambda^*_{\pi(h)}) \, for \, j \in N - \{\pi(h+1)\}.$$

□

We show that the condition $\lambda^*_{\pi(1)} = min\{\lambda^*_j : j \in N^{\neq}\} > 0$ is necessary in the previous corollary. Consider the following example.

Example 3 *Let* $x^* = (1, 0.3)$, $x' = (1, 0.1)$ *and* $x'' = (0.8, 0.9)$ *giving the corresponding values* $\lambda^*_1 = 0$ *and* $\lambda^*_2 = 0.25$. *So, we have* $l(\lambda^*_1) = \delta(x^*, (1, 0.1)) = (1, 0)$ *and* $l(\lambda^*_2) = \delta(x^*, (0.95, 0.3)) = (0, 1)$ *and hence* $d(l(\lambda^*_1), l(\lambda^*_2)) = 2$.

Corollary 4.2.8 *Corollary 4.2.7 remains valid in the case*

$$\lambda^*_{\pi(t)} = max\{\lambda^*_j : j \in N^{\neq}\} = 1.$$

Proof. It suffices to prove that the chosen tie breaking rule may be extended to the case $x^*_{\pi(t)} \in [0, 1]$. Since the equation $\lambda^*_{\pi(t)} = 1$ implies that $x^*_{\pi(t)} = x''_{\pi(t)}$ then $\delta(x^*_{\pi(t)}, x^*_{\pi(t)})$ may be defined as

$$\delta(x^*_{\pi(t)}, x^*_{\pi(t)}) = \begin{cases} x''_{\pi(t)} & \text{if } x''_{\pi(t)} \in \{0, 1\} \\ 1 & \text{if } x''_{\pi(t)} > x'_{\pi(t)} \\ 0 & \text{if } x''_{\pi(t)} < x'_{\pi(t)} \end{cases} \tag{4.14}$$

However, the case $x''_{\pi(t)} \in \{0,1\}$ is already contained in the other two cases (which represent the imposed tie breaking rule). Indeed, if $x''_{\pi(t)}$ is equal to 1, then $x''_{\pi(t)} > x'_{\pi(t)}$ (since $x''_{\pi(t)} \neq x'_{\pi(t)}$), which corresponds to the second case, while if $x''_{\pi(t)}$ is equal to 0, then $x''_{\pi(t)} < x'_{\pi(t)}$, which corresponds to the last case. So, the definition of $l_j(\lambda)$ given in equation (4.13) is valid in the case $\lambda^*_{\pi(t)} = 1$ as well as in the case given by Corollary 4.2.7. □

Another way to see the necessity of the condition $\lambda^*_{\pi(1)} = min\{\lambda^*_j : j \in N^{\neq}\} > 0$ is to note that is not possible to extend the imposed tie breaking rule to the case $x^*_{\pi(1)} \in [0,1]$. Indeed, since $\lambda^*_{\pi(1)} = 0$ compels $x^*_{\pi(1)} = x'_{\pi(1)}$ then $\delta(x^*_{\pi(1)}, x^*_{\pi(1)})$ may be defined as

$$\delta(x^*_{\pi(1)}, x^*_{\pi(1)}) = \begin{cases} x'_{\pi(1)} & \text{if } x''_{\pi(1)} \in \{0,1\} \\ 1 & \text{if } x''_{\pi(1)} > x'_{\pi(1)} \\ 0 & \text{if } x''_{\pi(1)} < x'_{\pi(1)} \end{cases} \tag{4.15}$$

Now, it is easy to check that the first case in equation (4.15) can not be omitted while the definition of $\delta(x^*_{\pi(1)}, x^*_{\pi(1)})$ remains in force, i.e., the first case in equation (4.15) is not contained in the remaining two cases.

However, if we change the imposed tie breaking rule such that

$$\delta(x^*_j, x^*_j) = \begin{cases} 1 & \text{if } x''_j < x'_j \\ 0 & \text{if } x''_j > x'_j \end{cases} \tag{4.16}$$

then it is easily demonstrated that Corollary 4.2.7 will be true if $\lambda^*_{\pi(1)} = min\{\lambda^*_j : j \in N^{\neq}\} = 0$, but will not be true if $\lambda^*_{\pi(t)} = max\{\lambda^*_j : j \in N^{\neq}\} = 1$. In other words, informally speaking, complementing the tie breaking rule corresponds to complementing the necessary conditions.

It should be emphasized that if we change the definition of directional rounding

(equation (4.17)) so that $\delta(x_j^*, x_j')$ is defined as

$$\delta(x_j^*, x_j') = \begin{cases} 0 & \text{if } x_j' < x_j^* \\ 1 & \text{if } x_j' > x_j^* \\ 0 \text{ or } 1 & \text{otherwise} \end{cases} \tag{4.17}$$

and if we suppose that we use the tie breaking rule from Corollary 4.2.7 for the last case, then Corollary 4.2.7 will hold when $\lambda^*_{\pi(1)} = 0$ or $\lambda^*_{\pi(t)} = 1$ and even when $\lambda^*_{\pi(1)} < 0$ or $\lambda^*_{\pi(t)} > 1$. The justification for this claim is based on the fact that in this case the components $l_j(\lambda^*_{\pi(h)})$ of any vector $l(\lambda^*_{\pi(h)})$ will be given by equation (4.13).

Lemma 4.2.9 *Let x' and x'' be focal points and let x^* be the base point, where $\lambda_j^* = \frac{x_j^* - x_j'}{x_j'' - x_j'}$ for $j \in N^{\neq}$. If $\lambda^*_{\pi(1)} < \lambda^*_{\pi(2)} < \cdots < \lambda^*_{\pi(h-1)} < \lambda^*_{\pi(h)} = \lambda^*_{\pi(h+1)} = \cdots = \lambda^*_{\pi(h+k)} < \cdots < \lambda^*_{\pi(t)}$, $k \geq 1$ then*

$$l_j(\lambda^*_{\pi(h-1)}) = l_j(\lambda^*_{\pi(h)}) = \cdots = l_j(\lambda^*_{\pi(h+k)}) \; for \; j \in N - \{\pi(h-1), \pi(h), \ldots, \pi(h+k)\}.$$

Moreover, if $\lambda^*_{\pi(1)} > 0$ and $\lambda^*_{\pi(t)} < 1$ and ties in directional rounding $\delta(x_j^*, x_j^*)$ which arise in the case $0 < x_j^* < 1$ are broken by setting

$$\delta(x_j^*, x_j^*) = \begin{cases} 1 & \text{if } x_j'' > x_j' \\ 0 & \text{if } x_j'' < x_j' \end{cases} \tag{4.18}$$

then

$$l_{\pi(h+p)}(\lambda^*_{\pi(h)}) = 1 - l_{\pi(h+p)}(\lambda^*_{\pi(h-1)}) \; for \; p = 0, 1, \ldots, k$$

Proof. First, note that the vectors $l(\lambda^*_{\pi(h)}), l(\lambda^*_{\pi(h+1)}), \ldots, l(\lambda^*_{\pi(h+k)})$ are the same since $\lambda^*_{\pi(h)} = \lambda^*_{\pi(h+1)} = \cdots = \lambda^*_{\pi(h+k)}$. If $x_j' = x_j''$ and $j \in N - \{\pi(h-1), \pi(h), \ldots, \pi(h+k)\}$ the conclusion of the lemma is immediate. On the other hand if $x_j' \neq x_j''$ and $\lambda^*_{\pi(j)} < \lambda^*_{\pi(h-1)} < \cdots < \lambda^*_{\pi(h+k)}$ or $\lambda^*_{\pi(j)} > \lambda^*_{\pi(h+k)} > \cdots > \lambda^*_{\pi(h)}$ then the definition of the vectors $l(\lambda)$ implies that $l_j(\lambda^*_{\pi(h-1)}) = l_j(\lambda^*_{\pi(h)}) = \cdots = l_j(\lambda^*_{\pi(h+1)}) = l_j(\lambda^*_{\pi(h+k)})$. However, by Corollary 4.2.7 which establishes that $l_{\pi(j+1)}(\lambda^*_{\pi(j+1)}) = 1 - l_{\pi(j)}(\lambda^*_{\pi(j)})$ for

each pair $(\pi(h+p), \pi(h-1))$, for $p = 0, 1, \ldots, k$ we obtain

$$l_{\pi(h+p)}(\lambda^*_{\pi(h+p)}) = 1 - l_{\pi(h+p)}(\lambda^*_{\pi(h-1)}) \text{ for } p = 0, 1, \ldots, k.$$

which is equivalent to

$$l_{\pi(h+p)}(\lambda^*_{\pi(h)}) = 1 - l_{\pi(h+p)}(\lambda^*_{\pi(h-1)}) \text{ for } p = 0, 1, \ldots, k.$$

□

In the same way as in Corollary 4.2.7 it can be shown that if $0 < \lambda^*_{\pi(1)} < \lambda^*_{\pi(2)}, \ldots, \lambda^*_{\pi(h-1)} < \lambda^*_{\pi(h)} = \lambda^*_{\pi(h+1)} = \cdots = \lambda^*_{\pi(h+k)} < \ldots, < \lambda^*_{\pi(t)} \leq 1$ and if ties are broken as proposed in Corollary 4.2.7 (and Lemma 4.2.9) then

$$l_j(\lambda^*_{\pi(h+k+1)}) = l_j(\lambda^*_{\pi(h)}) \, for \, j \in N - \{\pi(h+k+1)\}$$

$$l_{\pi(h+k+1)}(\lambda^*_{\pi(h+k+1)}) = 1 - l_{\pi(h+k+1)}(\lambda^*_{\pi(h)}).$$

In order to produce a larger number of solutions and therefore to increase the chance to reach an optimal solution we can treat $\lambda^*_{\pi(h+p)}$ for $p = 0, 1, \ldots, k$ as distinct values and associate to each of them the vectors defined as

$$l_j(\lambda^*_{\pi(h+p)}) = \begin{cases} l_j(\lambda^*_{\pi(h+p-1)}) & \text{if } j \neq \pi(h+p) \\ 1 - l_j(\lambda^*_{\pi(h+p-1)}) & \text{if } j = \pi(h+p) \end{cases} \tag{4.19}$$

The vector $l(\lambda^*_{\pi(h+k)})$ then corresponds to the vector $l(\lambda^*_{\pi(h)})$ defined in the previous lemma.

4.2.4 Exploiting the results to generate Star Paths

To generate a Star Path $L(x^*, x', x'')$ where $Line(x', x'')$ is restricted to the segment $[x', x'']$ we propose the following Algorithm 43 which exploits the previously proven statements. Since we consider just the segment $[x', x'']$, we want to produce a set of vectors obtained by directional rounding using points which belong to the segment as focal points and x^* as base point, i.e., $L(x^*, x', x'') = \{l(\lambda) = \delta(x^*, x) : x = x' + \lambda(x'' - x'),\ 0 \leq \lambda \leq 1\}$. According to Theorem 4.2.5 the Star Path $L(x^*, x', x'')$ can be constructed by rounding

directionally only a finite number of points from the segment $[x', x'']$, i.e., $L(x^*, x', x'') = \{l(\lambda) : \lambda \in \{0, 1\} \text{ or } \lambda = \lambda_j^*, \text{ with } 0 \leq \lambda_j^* \leq 1, \ j \in N^{\neq}\}$. Further as shown in the remaining statements there are benefits caused by sorting λ_j^*, so we assume that all λ_j^* with values between 0 and 1 are sorted in non-decreasing order, i.e., $0 \leq \lambda_{\pi(1)}^* \leq \lambda_{\pi(2)}^* \leq \cdots \leq \lambda_{\pi(p)}^* \leq 1$. In the case that $\lambda_{\pi(1)}^* > 0$, we can add an artificial $\lambda_{\pi(0)}^*$ with value 0 in order to cover case $\lambda = 0$ (note in this case $\pi(0)$ does not correspond to any index). On the other hand if $\lambda_{\pi(p)}^* < 1$ there is no need to extend the array of λ_j^* values since according to Lemma 4.2.5 $l(1) = l(\lambda_{\pi(p)}^*)$. Hence, without loss of generality we may suppose that $\lambda_{\pi(0)}^* = 0$. Based on these observations, the generation of the Star Path $L(x^*, x', x'')$ may be constructed in the following way. The first vector to be generated is the vector $l(\lambda_{\pi(0)}^*)$ which is obtained using the definition given in Lemma 4.2.5 together with the tie breaking rule. From then on, each vector is derived from the immediately preceding vector using Corollary 4.2.7, Corollary 4.2.8 and Lemma 4.2.9 in the following manner. Let $N' = \{j \in N^{\neq} : 0 \leq \lambda_j^* \leq 1\}$, $n' = |N'|$ and let $N^0 = \{j \in N^{\neq} : \lambda_j^* = 0\}$, $n^0 = |N^0|$. Then the vector $l(\lambda_{\pi(n^0+1)}^*)$ may be obtained from the vector $l(\lambda_{\pi(0)}^*)$ by adjusting values at the positions $\pi(1), \ldots, \pi(n^0)$ according to the rules for generating vector $l(\lambda_{\pi(n^0+1)}^*)$. In order to avoid (possible) large distances between vectors $l(\lambda_{\pi(0)}^*)$ and $l(\lambda_{\pi(n^0+1)}^*)$ we may generate additional vectors $l(\lambda_{\pi(i)}^*)$, $i = 1, \ldots, n'$ obtained from the previous vector by adjusting entries at the position $\pi(i)$ according to the rules for generating vector $l(\lambda_{\pi(n^0+1)}^*)$. Then each element $l(\lambda_{\pi(h)}^*)$, $h > n^0$ can be derived directly from the vector $l(\lambda_{\pi(h-1)}^*)$, by just complementing the value at the position $\pi(h)$ (Corollary 4.2.7, Corollary 4.2.8 and Lemma 4.2.9). Note that all $\lambda_{\pi(h)}^*$, $h > n^0$, are treated as distinct in order to increase the number of generated

vectors as explained above.

Algorithm 43: Generating Star Path $L(x^*, x', x'')$

Function `Star Path`(x^*, x', x'')

Compute $\lambda_j^* = \frac{x_j^* - x_j'}{x_j'' - x_j'}$ for each $j \in N^{\neq} = \{j \in N : x_j' \neq x_j''\}$;

Let $N' = \{j \in N^{\neq} : 0 \leq \lambda_j^* \leq 1\}$, and let $N^0 = \{j \in N^{\neq} : \lambda_j^* = 0\}$, set $n' = |N'|$ and $n^0 = |N^0|$;

Sort all λ_j^* for $j \in N'$ in nondecreasing order, i.e., so that $\lambda_{\pi(1)}^* \leq \lambda_{\pi(2)}^* \leq \cdots \leq \lambda_{\pi(n')}^*$;

Set $\lambda_{\pi(0)}^* = 0$;

Determine $x = l(\lambda_{\pi(0)}^*) = \delta(x^*, x')$ using Lemma 4.2.5;

Set $L = \{x\}$;

for $j = 1$ *to* n^0 **do**

  $x_{\pi(j)} = l_{\pi(j)}(\lambda_{\pi(n^0+1)}^*)$;

  $L = L \cup \{x\}$;

end

for $j = n^0 + 1$ *to* n' **do**

  $x_{\pi(j)} = 1 - x_{\pi(j)}$;

  $L = L \cup \{x\}$;

end

return L;

Theorem 4.2.2 *Given x' and x'' as focal points and x^* as base point, $\delta(x^*, [x', x'']) = \delta(x^*, [y, x''])$ where $y \in Ray(x^*, x') - \{x^*\}$. Furthermore, $\delta(x^*, [x', x'']) = \delta(x^*, [x', z])$ where $z \in Ray(x^*, x'') - \{x^*\}$.*

Proof. Let y be any point on the ray from x^* through x' different from x^*. Hence the vector y will be represented as $y = x^* + \alpha(x' - x^*)$ with $\alpha > 0$. According to Lemma 4.2.4, we have $\delta(x^*, x') = \delta(x^*, y)$. So to show that $\delta(x^*, [x', x'']) = \delta(x^*, [y, x''])$ it suffices to show that the value of $\lambda_j^* = \frac{x_j^* - x_j'}{x_j'' - x_j'}$ values sorted in nondecreasing order retains this order by replacing x_j' with y_j. Indeed, from Algorithm 43 it follows that each Star Path is uniquely determined by the starting point of line segment and the order of λ_j^* values. So,

let $\lambda'^* = \frac{x_j^* - y_j}{x_j'' - y_j}$ and suppose we have the order $0 \leq \lambda_1^* \leq \lambda_2^* \leq \cdots \leq \lambda_{n'}^* \leq 1$. Then, for each j, $\lambda_j'^*$ may be written as $\lambda_j'^* = \frac{\alpha(x_j^* - x_j')}{x_j'' - x_j^* + \alpha(x_j^* - x_j')}$ or equivalently as $\lambda_j'^* = \frac{\alpha(x_j^* - x_j')}{x_j'' - x_j' + (\alpha-1)(x_j^* - x_j')}$. Now, from the definition of the λ_j^* values and the last expression of $\lambda_j'^*$ we have $\lambda_j'^* = \frac{\alpha\lambda_j^*}{1 - \lambda_j^* + \alpha\lambda_j^*}$. Next, it is easy to check that $\lambda_j'^* \leq \lambda_k'^* \Leftrightarrow \lambda_j^* \leq \lambda_k^*$. So,

$$0 \leq \lambda_1^* \leq \lambda_2^* \leq \cdots \leq \lambda_{n'}^* \leq 1 \Leftrightarrow 0 \leq \lambda_1'^* \leq \lambda_2'^* \leq \cdots \leq \lambda_{n'}'^* \leq 1$$

and therefore $\delta(x^*, [x', x'']) = \delta(x^*, [y, x''])$.
In a similar way, if we choose any point $z \neq x^*$ on the ray from x^* through x', i.e., $z = x^* + \alpha(x'' - x^*)$ with $\alpha > 0$, it holds $\delta(x^*, x'') = \delta(x^*, z)$ (according to Lemma 4.2.4). Further if $\lambda''^* = \frac{x_j^* - z_j}{x_j'' - z_j}$, we obtain, analogously to the previous case, $\lambda_j''^* = \frac{\lambda_j}{\lambda_j + \alpha}$ for each j. Then it follows that $\lambda_j''^* \leq \lambda_k''^* \Leftrightarrow \lambda_j^* \leq \lambda_k^*$ and therefore $\delta(x^*, [x', x'']) = \delta(x^*, [x', z])$. □

As a consequence of Theorem 4.2.2, we have the following result.

Corollary 4.2.10 *Given x' and x'' as focal points and x^* as a base point, $\delta(x^*, [x', x'']) = \delta(x^*, [y, z])$ where $y \in Ray(x^*, x') - \{x^*\}$ and $z \in Ray(x^*, x'') - \{x^*\}$.*

□

This corollary effectively says that $\delta(x^*, [x', x'']) = \delta(x^*, Cone(x^*, \{x', x''\}) - \{x^*\})$.

Theorem 4.2.3 *Given x^* as base point and X as a set of focal points, if $\delta(x^*, X) = x'$ then $\delta(x^*, conv(X)) = x'$.*

Proof. If $y \in conv(X)$ then there exists a set of points $x^1, \ldots, x^p$ such that $y = \sum_{i=1}^p \alpha_i x^i$ and $\sum_{i=1}^p \alpha_i = 1$ with $\alpha_i \geq 0$ and $x^i \in X$ for $i = 1, \ldots, p$. It is easy to see that we have $y_j \geq x_j^*$ if $x_j^i \geq x_j^*$ for all $i = 1, \ldots, p$ and $y_j \leq x_j^*$ if $x_j^i \leq x_j^*$ for all $i = 1, \ldots, p$. So from the definition of directional rounding δ, if we assume that "tie breaking" rule is the same for all solutions x^i $(i = 1, \ldots, p)$ when $x_j^i = x_j^*$ with $0 < x_j^* < 1$ for all $j = 1, \ldots, n$, we have

$$\delta(x_j^*, y_j) = \delta(x_j^*, x_j^i).$$

□

4.3 Fundamental Analysis of Star Paths with Directional Rounding

4.3.1 Standard LP basic solution representation

The bounded simplex method proposed by Dantzig [57, 58] is an efficient method to solve the LP- relaxation of the MIP problem by systematically exploring extreme points of a solution space. The search for an optimal extreme point is performed by pivot operations, each of which moves from one extreme point to an adjacent extreme point by removing one variable from the current basis and bringing another variable (which is not in the current basis) into the basis. For our purposes, the procedure can be depicted in the following way. Suppose that the method is currently at some extreme point $x(0)$ with corresponding basis B. The set of indices of all other variables (nonbasic variables) will be designated with $\overline{B} = N - B$. The extreme points adjacent to $x(0)$ have the form

$$x(j) = x(0) - \theta_j D_j \text{ for } j \in \overline{B} \tag{4.20}$$

where D_j is a vector associated with the nonbasic variable x_j, and θ_j is the change in the value of x_j that moves the current solution from $x(0)$ to $x(j)$ along their connecting edge. The LP basis representation identifies the components D_{kj} of D_j, as follows

$$D_{kj} = \begin{cases} ((A^B)^{-1}A)_{kj} & \text{if } k \in B \\ \xi & \text{if } k = j \\ 0 & \text{if } k \in \overline{B} - \{j\} \end{cases} \tag{4.21}$$

where A^B represents matrix obtained from matrix A by selecting columns that correspond to the basic variables and $\xi \in \{-1, 1\}$. We choose the sign convention for entries of D_j that yields a coefficient for x_j of $D_{jj} = 1$ if x_j is currently at its lower bound at the vertex $x(0)$, and of $D_{jj} = -1$ if x_j is

currently at its upper bound at $x(0)$. We assume throughout the following that $x(0)$ is feasible for the LP problem, though this assumption can be relaxed. In general, we require that $x(0)$ is a basic solution (feasible or not) – i.e., $x(0)$ is a feasible extreme point for a region that results by discarding some of the constraints that define the original LP feasible region.

Note that if we consider an extreme point $x(0)$ and its adjacent extreme points $x(j)$ for $j \in \overline{B}$, we can conclude that the points $x(j)$ for $j \in \overline{B}$ are linearly independent and that the point $x(0)$ does not belong to the plane spanned by these points. Furthermore, this observation holds even when these θ_j values are replaced by any positive value.

4.3.2 Justification of Star Paths with Directional Rounding

Lemma 4.3.1 *Let $x(0)$ be a basic extreme point associated with a basis B, and define $x(j)$ for $j \in \overline{B}$ by (4.20) for any given positive value θ_j^* for θ_j, i.e.,*

$$x(j) = x(0) - \theta_j^* D_j, j \in \overline{B} \tag{4.22}$$

Then, we obtain:

$$Cone(x(0), X(\overline{B})) = \{x(0) - \sum_{j \in \overline{B}} \lambda_j D_j : \lambda_j \geq 0, j \in \overline{B}\}. \tag{4.23}$$

Proof. The proof follows directly from the definition of the cone. □

Lemma 4.3.2 *Let $x(0)$ be a basic extreme point with its associated basis denoted by B. Then all feasible solutions of the LP problem belong to the $Cone(x(0), X(\overline{B}))$.*

Proof. Without loss of generality we can assume that $B = \{1, 2, \ldots m\}$, where m represents the number of rows in the matrix A and let $\mathcal{B} = (A^B)^{-1}$. Therefore, each solution x may be represented as $x = [x_B, x_{\overline{B}}]^T$. Furthermore, each LP–feasible solution x can be expressed as $x_B = \mathcal{B}b - \mathcal{B}A^{\overline{B}}x_{\overline{B}}$. This last equality may be rewritten as $x_B = \mathcal{B}b - [\mathcal{B}A^{m+1}, \ldots \mathcal{B}A^n]x_{\overline{B}}$.

Defining a new set of variables $\theta_{\overline{B}} = x(0)_{\overline{B}} - x_{\overline{B}}$ the last equation becomes $x_B = \mathcal{B}b - [\mathcal{B}A^{m+1}, \ldots \mathcal{B}A^n]x(0)_{\overline{B}} + [\mathcal{B}A^{m+1}, \ldots \mathcal{B}A^n]\theta_{\overline{B}}$ or equivalently $x_B = x(0)_B + [\mathcal{B}A^{m+1}, \ldots \mathcal{B}A^n]\theta_{\overline{B}}$. Using the last equation we obtain the following representation of the solution x:

$$x = x(0) - \begin{bmatrix} \mathcal{B}A^{m+1} & \mathcal{B}A^{m+2} & \ldots & \mathcal{B}A^n \\ \xi & 0 & \ldots & 0 \\ 0 & \xi & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \xi \end{bmatrix} \theta_{\overline{B}} \tag{4.24}$$

However adjusting entries of matrix in (4.24) according to replacing $\theta_{\overline{B}}$ with $|\theta_{\overline{B}}|$ yields a matrix whose columns are directional vectors $D_j, j \in \overline{B}$, and therefore $x = x(0) - \sum_{j\in\overline{B}} D_j|\theta_j|$, i.e., $x \in Cone(x(0), X(\overline{B}))$. Since x is an arbitrarily chosen feasible point we conclude that all feasible points belong to the $Cone(x(0), X(\overline{B}))$. □

Corollary 4.3.3 *All MIP feasible solutions belong to the $Cone(x(0), X(\overline{B}))$.*

Lemma 4.3.4 *Let $x(0)$ be an LP feasible extreme point with associated basis B, then $x(0)$ does not belong to $Half\_Space(X(\overline{B}))$, but belongs to the complementary half space.*

Proof. By definition, we have $Plane(X(\overline{B})) = \{x(0) - \sum_{j\in\overline{B}} \lambda_j D_j : \sum_{j\in\overline{B}} \lambda_j = 1\}$. So, its corresponding half space is $Half\_Space(X(\overline{B})) = \{x(0) - \sum_{j\in\overline{B}} \lambda_j D_j : \sum_{j\in\overline{B}} \lambda_j \geq 1\}$. Keeping in mind the linear independence of vectors D_j, it follows that $Half\_Space(X(\overline{B}))$ does not contain point $x(0)$. So, $x(0)$ belongs to the complementary half space. □

Lemma 4.3.5 *Let $x(0)$ be an LP feasible extreme point with the associated basis B. Then all optimal MIP solutions, excluding $x(0)$, belong to $Half\_Space(X(\overline{B}))$.*

Proof. The lemma follows directly from the fact that $Half\_Space(X(\overline{B}))$ may be considered as a valid cutting plane that excludes $x(0)$ as feasible. □

Corollary 4.3.6 *The previous two lemmas remain true if the set $X(\overline{B})$ is replaced by the set $\{x(0) - \theta_j^* D_j : j \in \overline{B},\ 0 < \theta_j^* < \theta_j\}$.*

Theorem 4.3.1 *Let $x(0)$ be an LP feasible extreme solution with associated basis B, and let $X(\overline{B}) = \{x(0) - \theta_j^* D_j : j \in \overline{B}\}$ where θ_j^* is any given positive value. Then for any MIP feasible solution x' there is a convex region $X \subset Face(X(\overline{B}))$ such that $\delta(x(0), x) = x'$ for all $x \in X$. Moreover, if X is not polyhedral, there is a polyhedral subset of X for which this conclusion is true.*

Proof. Given that all MIP feasible solutions are in the $Cone(x(0), X(\overline{B})) = \{x(0) - \sum_{j\in\overline{B}} \lambda_j D_j : \lambda_j \geq 0\}$, it follows that for each MIP feasible solution x' the $Ray(x(0), x')$ which belongs to that cone intersects the $Face(X(\overline{B}))$. Denote this intersection point by y, hence $y = Ray(x(0), x') \cap Face(X(\overline{B}))$. According to the previous lemmas the point y satisfies $\delta(x(0), y) = \delta(x(0), x') = x'$. The last condition means that for any MIP feasible solution x', there is at least one solution which belongs to the $Face(X(\overline{B}))$ and which produces the solution x' by directional rounding relative to the base solution $x(0)$. However, if there exists more than one solution which may be directionally rounded to the yield x', then according to Theorem 4.2.3 the directional rounding of any solution in the convex hull X of these points, produces the same solution x'. Further, if X is not polyhedral, its polyhedral subset can be identified as a set of all convex combinations of a finite number of points from X. □

Corollary 4.3.7 *The previous theorem is also valid when the word "feasible" is replaced by "optimal".*

The next result has been proven in Chapter 1, but we repeat it to facilitate the proof of Theorem 4.3.2 bellow.

Lemma 4.3.8 *An optimal solution for the 0–1 MIP problem may be found at an extreme point of the LP feasible set.*

Theorem 4.3.2 *Let $x(0)$ be an LP optimal extreme point and let C be a valid cutting plane for the problem MIP, that excludes $x(0)$ and is satisfied by all MIP feasible solutions. Then, if the problem MIP has an optimal solution there exists a polyhedral region $P \subset C$ such that each MIP feasible solution can be obtained by directional rounding of some point in P. Moreover, at*

least one optimal MIP solution is obtained by directional rounding of some extreme point of P.

Proof. Since cutting plane C separates $x(0)$ from all MIP feasible solutions, each ray starting at $x(0)$ through some MIP feasible solution, say x, intersects the plane C at some point, say x', i.e., $x' = Ray(x(0), x) \cap C$. However, Lemma 4.2.4 demonstrates that directional rounding of x' gives the point x. Therefore each MIP feasible solution can be obtained by rounding some point from the region $P = conv(\{Ray(x(0), x) \cap C : x\ MIP\ feasible\ solution\})$. This set P is a polyhedral region since it is a convex hull of a finite number of MIP feasible solutions. Hence, P may be rewritten by reference to the convex hull of its extreme points $x'^1, x'^2, \ldots, x'^p$ as $P = conv(\{x'^i : i = 1, \ldots, p\})$. If X is the MIP feasible set, then $X \subseteq \delta(x(0), P)$.
Assume that no one of the solutions $x^i = \delta(x(0), x'^i)$ for $i = 1, \ldots, p$, is MIP optimal. Therefore, using Lemma 4.3.8, we can prove that $x(0)$ and an optimal solution for x^{opt} MIP are on different sides of plane $H = \{\sum_{i=1}^{p} \lambda_i x^i : \sum_{i=1}^{p} \lambda_i = 1\}$ (see Figure 4.2). (If we suppose $x(0)$ and x^{opt} are on the same side then x^{opt} is inside the truncated cone, defined by rays from $x(0)$ through points x'^i and H, contradicting the fact that x^{opt} is an extreme point.) So, there is a point $y \in H$ such that $y = x(0) + \lambda(x^{opt} - x(0))$, $0 < \lambda < 1$. Using the linearity of the objective function and the optimality of the LP-solution $x(0)$ we conclude that $cx(0) \leq cy \leq cx^{opt}$. On the other hand, since y belongs to H it can be represented as $\sum_{i=1}^{p} \lambda_i x^i$, $\sum_{i=1}^{p} \lambda_i = 1$, so $cy = \sum_{i=1}^{p} \lambda_i cx^i > cx^{opt}$ because none of the points x^i is MIP optimal. This contradicts our starting assumption that none of the points x^i is MIP optimal. Hence, at least one solution among solutions $x^i = \delta(x(0), x'^i)$ for $i = 1, \ldots, p$ is an optimal MIP solution and consequently can be obtained by directional rounding of some extreme point of P. □

Corollary 4.3.9 *The previous lemmas and theorems hold when $x(0)$ is any basic solution (feasible or not) with associated basis B.*

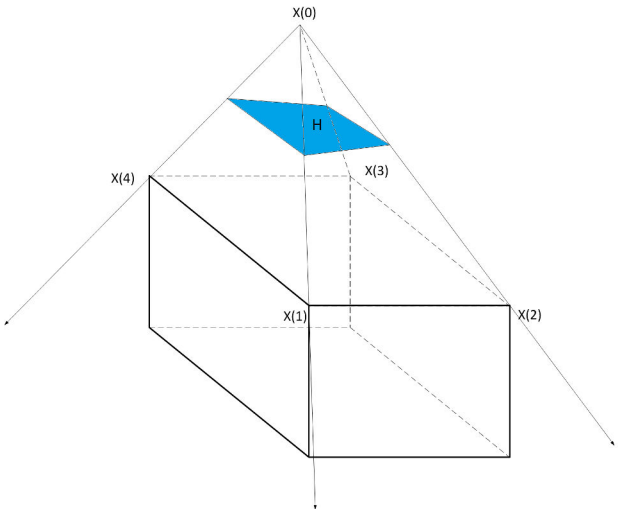

Figure 4.2: Cone $C(x(0), X(\bar{B}))$ and valid cutting plane C.

4.4 Convergent Scatter Search with directional rounding

We now build on the fact that Scatter Search consists of a systematic exploration of a solution space relative to a set of reference points. As pointed out, these points typically consist of good solutions obtained by prior problem solving effort, where the term "good solution" refers not only to solutions with good objective function values, but also to solutions which increase the diversity of the set of reference points.

4.4.1 Variant of Convergent Scatter Search

In this section we propose a version of scatter search with directional rounding that converges in a finite number of iterations to an optimal solution for the 0–1 MIP problem. The resulting Convergent Scatter Search algorithm is justified by the preceding theorems. Let $x(0)$ be an optimal solution of

the LP-relaxation with the associated basis B and let $\pi x \leq \pi_0$ be a valid cutting plane for problem MIP that excludes $x(0)$ without excluding any MIP feasible solutions. Valid cutting planes are very well studied in the literature. There are several procedures to generate deep cuts including convexity cuts or intersection cuts (see for example, [5, 8, 10, 89, 215, 228]). Recently, Balas and Margot [7] introduced a generalization of the intersection cuts that dominate the original ones. We define the polyhedral region P as the intersection of the hyperplane $\pi x = \pi_0$ with the set of feasible solutions of the LP-relaxation denoted by $\overline{X}$, i.e., $P = \{x : \pi x = \pi_0\} \cap \overline{X}$. By the preceding results we know that an optimal MIP solution belongs to the set $\delta(x(0), P)$. The Convergent Scatter Search algorithm works as follows. At each iteration, we choose an extreme point of the current polyhedron (polyhedral cone) P, say x' and select a cone $C(x')$ originating at x' such that is easy to compute the finite set $\delta(x(0), C(x'))$. The process is repeated on the new polyhedron $P = P - C(x')$ until the volume of the current polyhedron P is null. Thus, the algorithm successively nibbles away portions $C(x')$ starting from the corner x' each time until nothing is left (see Algorithm 44).

Algorithm 44: Framework of the convergent star path algorithm $CSP()$

Function `CSP()`;

1 Let $x(0)$ be an optimal solution of LP-relaxation;
2 Set $P = \{x : \pi x = \pi_0\} \cap \overline{X}$, where $\pi x \leq \pi_0$ is a cutting plane;
3 **while** $Volume(P) > 0$ **do**
4 Choose any extreme point of P, i.e., x';
5 Construct the cone $C(x')$ originating at x';
6 Compute $S = \delta(x(0), C(x'))$;
7 Update the best solution $x^* = \arg\min_{x \in S \cup x^*} cx$;
8 Reduce the current polyhedral $P = P - C(x')$;

end

9 **Return** x^*;

Now, we explain how we construct at each iteration the cone $C(x')$ originating at an extreme point x' of the current polyhedron P. Let B denote

the basis associated with the extreme point x' relative to the polyhedron P. For each adjacent point of x', $x'(h)$, $h \in \overline{B}$, we determine $y'(h)$ belonging to the $Ray(x', x'(h))$ such that directional rounding of all points from open sub-edge $]x', y'(h)[$ yields the same 0–1 solution. These points $y'(h)$ can be computed using Theorem 4.2.1 or the first iteration of Algorithm 43. More precisely, $y'(h) = x' + \lambda^*(x'(h) - x')$ with $\lambda^* = min\{\lambda_j^* : 0 < \lambda_j^* = \frac{x(0)_j - x'_j}{x'(h)_j - x'_j} \leq 1\}$. The constructed cone originating at the extreme point x' is defined as $C(x') = Cone(x', Y) \cap H^-$ where $Y = \{y'(h) : h \in \overline{B}\}$ and $H^- = \{\sum_{h \in \overline{B}} \lambda_h y'(h) : \sum_{h \in \overline{B}} \lambda_h < 1\}$.

A Convergent Scatter Search algorithm may be implemented in two different ways according to whether or not we reduce the size of the LP polyhedron at the same time that we reduce the size of $Plane(X(\overline{B}))$. These two different algorithmic approaches are depicted as Algorithm 45 and Algorithm 46 respectively. Both start by solving the LP relaxation and creating $Plane(X(\overline{B}))$ as a convex hull of all extreme points adjacent to an optimal solution of the LP relaxation. Then both methods choose the best extreme point x' (the point that minimizes the objective function of the initial problem) which belongs to the set $X(\overline{B})$ and determine the cone $C(x')$ to be directionally rounded. Then the truncated cone $C(x')$ is directionally rounded using Theorem 4.4.2. The first algorithm then continues to repeat overall process on $Plane(X(\overline{B})) - C(x')$, while the second algorithm repeats the overall process on the new MIP problem derived from the initial problem by imposing the valid cut which excludes the optimal solution of LP relaxation and the pseudo cut which eliminates extreme point x'. The pseudo cut is determined as a hyperplane passing through points from the set Y and the optimal solution of the LP relaxation (which are linearly independent). The first procedure stops when the difference between the objective function values at x' and the best encountered integer solution is less than ϵ while the second stops when the difference between the value of the LP relaxation of the new created problem and the value of best integer solution found is less than ϵ. (If all the data are integer the value of ϵ can be set to 1.) The coefficients of the hyperplane containing $Plane(X(\overline{B}))$, or the coefficients of the hyperplane passing through points from the set Y and $x(0)$, may be easily determined

as a feasible non-zero solution of a linear program with a trivial objective function subject to $\pi x = \pi_0$ for $x \in X(\overline{B})$ or $x \in Y \cup \{x(0)\}$.

Algorithm 45: Convergent Star Path algorithm -version 1 $CSP1()$

Function `CSP1()`

Let $x(0)$ be an optimal solution of LP-relaxation;
Set $P = \{x : \pi x = \pi_0\} \cap \overline{X}$, where $\pi x \leq \pi_0$ is a cutting plane;
repeat
  Let x' be an optimal solution of $min\{cx : x \in P\}$;
  Construct the cone $C(x')$ originating at x';
  Compute $S = \delta(x(0), C(x'))$;
  Update the best solution $x^* \leftarrow \arg\min_{x \in S \cup x^*} cx$;
  Reduce the current polyhedral $P = P - C(x')$;
until $cx^* - cx' \geq \epsilon$;
return x^*;

Algorithm 46: Convergent Star Path algorithm -version 2 $CSP2()$

Function `CSP2()`

repeat
  Let $x(0)$ be an optimal solution of LP-relaxation, i.e., $cx(0) = min\{cx : x \in \overline{X}\}$;
  Set $P = \{x : \pi x = \pi_0\} \cap \overline{X}$, where $\pi x \leq \pi_0$ is a cutting plane;
  Let x' be an optimal solution of $min\{cx : x \in P\}$;
  Construct the cone $C(x')$ originating at x';
  Compute $S = \delta(x(0), C(x'))$;
  Update the best solution $x^* \leftarrow \arg\min_{x \in S \cup x^*} cx$;
  Compute hyperplane $H = \{x : hx = h_0\}$ as one passing through points from the set Y and $x(0)$;
  Define H^+ as a half space bounded with H that does not contain x';
  Reduce the current LP polyhedral $\overline{X} = \overline{X} \cap P \cap H^+$;
until $cx^* - cx(0) \geq \epsilon$;
return x^*;

Implementation of the Convergent Scatter Search procedure requires a tool for enumerating all adjacent extreme points of a given extreme point

of a polyhedron. In some cases, if there is no degeneracy, enumeration of adjacent extreme points can be performed easily using the formulas of (4.20). Unfortunately, even if there is no degeneracy at the starting polyhedron of the LP relaxation degeneracy will appear during execution of the Convergent Scatter Search procedure since at each iteration we add some cutting plane to the starting LP problem. In consequence, we are unable to enumerate all extreme points using just formulas (4.20). Additionally, the Convergent Scatter Search procedure requires a lot of memory for rounding cone $C(x')$, making it unsuitable for large scale problems. In section 4.6, we propose heuristics based on the Convergent Scatter Search procedure.

4.4.2 Convergence proof of scatter search

To the best of our knowledge, up to now convergence results have been obtained for only a few metaheuristics [24, 98, 109]. In this subsection, we provide background and the convergence proof of the Scatter Search and Star Paths with Directional Rounding algorithms described in the previous subsection. More precisely, we provide theorems that demonstrate how to construct the truncated cone $C(x')$ originating at x' and how to directionally round it.

Theorem 4.4.1 *Let H be any hyperplane such that $H \cap C(x') \neq \emptyset$ and $H \cap conv(Y) = \emptyset$. Then $\delta(x(0), C(x')) = \delta(x(0), H \cap C(x'))$.*

Proof. By the definition of directional rounding, the fact that the directional rounding of multiple points relative to the base point $x(0)$ can yield the same 0–1 solution implies that those points belong to the same sub-space bounded by the hyperplanes $x_j = x(0)_j$. For example, in two dimensions if the directional rounding of some point A is equal to point (1,1) (i.e., $\delta(x(0), A) = (1, 1)$), then A belongs to the sub-space $x_1 \geq x(0)_1,\ x_2 \geq x(0)_2$. In turn, this observation implies that in the case of directionally rounding a line (rather than a single point), the points that lie on two consecutive intersections of the line and a hyperplane $x_j = x(0)_j$ will be rounded directionally to the same 0-1 point. It is easy to check that the set $\{H \cap C(x') : H \text{ a hyperplane},\ H \cap C(x') \neq \emptyset,\ H \cap conv(Y) = \emptyset\}$ equals

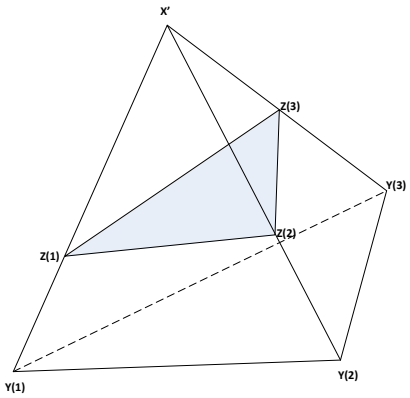

Figure 4.3: Cone C(x') and points z(h)

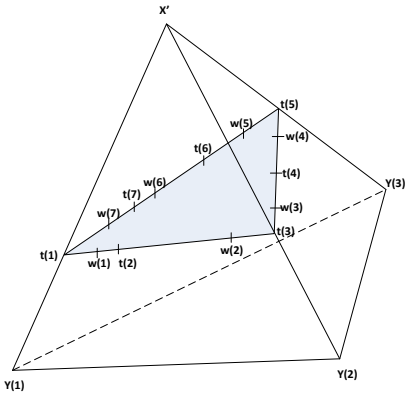

Figure 4.4: Cone C(x') and points from set T'

to $C(x')$. Hence, to prove the theorem it is enough to show that each set $H \cap C(x')$ is intersected by the same hyperplanes $x_j = x(0)_j$. However, no line segment $]x', y'(h)[$ is cut by a hyperplane $x_j = x(0)_j$ (otherwise, there will be at least two points on some segment $]x', y'(h)[$ which yield two different 0-1 solutions by directional rounding). If some set $H \cap C(x')$ is intersected by some hyperplane $x_l = x(0)_l$ then this hyperplane passes through point x' and therefore hyperplane $x_l = x(0)_l$ must intersect all sets $H \cap C(x')$ such that H is a hyperplane, $H \cap C(x') \neq \emptyset$, $H \cap conv(Y) = \emptyset$. So, for each hyperplane H such that $H \cap C(x') \neq \emptyset$ and $H \cap conv(Y) = \emptyset$, it follows that $\delta(x(0), C(x')) = \delta(x(0), H \cap C(x'))$. □

As consequence of the preceding theorem, we conclude that to directionally round the truncated cone $C(x')$ it suffices to round the set of points defined by the intersection of the cone $C(x')$ and any hyperplane H disjoint with $conv(Y)$. In other words it is possible to reduce the dimension of the set which is directionally rounded, i.e., instead of rounding $C(x')$ it suffices to round the lower dimensional set $H \cap C(x')$. Furthermore, the set $H \cap C(x')$ may be directionally rounded very efficiently as we show in the next theorem.

Theorem 4.4.2 *Let H be any hyperplane such that $H \cap C(x') \neq \emptyset$ and $H \cap conv(Y) = \emptyset$. Then the set $H \cap C(x')$ can be rounded by rounding a finite number of line segments lying inside the hyperplane H.*

Proof. For two points x' and x'', we define $\Delta(x', x'') = \{x(\lambda_j^*) : x(\lambda_j^*) = x' + \lambda_j^*(x'' - x'), 0 \leq \lambda_j^* = \frac{x(0)_j - x'_j}{x''_j - x'_j} \leq 1\}$. Let $z(h)$ denotes the point obtained by the intersection of the hyperplane H with the edge $[x', y'(h)]$, i.e., $z(h) = [x', y'(h)] \cap H$. Further, let us denote by Z the set of all $z(h)$ points, i.e., $Z = \{z(h) : h = 1, \ldots, |\overline{B}|\}$ (see Figure 4.3), and define the set of points $T = Z \cup \{\Delta(z(h), z(h')) : z(h)$ and $z(h')$ are adjacent points$\}$. Let us index points in the set T yielding $T = \{t(1), \ldots, t(p)\}$ where $p = |T|$. Then, we determine new points to be added to the set T, by randomly choosing points on each extreme edge according to the following rule. Let $[z(h), z(h')]$ be an edge and $t'(1), \ldots, t'(m)$ points on that edge determined by computing $\Delta(z(h), z(h'))$ such that any open sub-edge $]t'(k), t'(k+1)[$, $k = 1, \ldots m-1$, does not contain any other $t'(j)$ point. Then the points from each edge to be added are chosen as random points $w^k \in]t'(k), t'(k+1)[$ for $k = 0, \ldots, m$ where $t'(0) = z(h)$ and $t'(m+1) = z(h')$. Adding these points to set T the new set denoted T' is constituted (see Figure 4.4), i.e., $T' = T \cup \{w^k : k = 0, \ldots, m+1\}$ where $\{w^k : k = 0, \ldots, m+1\}$ represents set of all w^k points with respect to all edges. Therefore, we obtain $\delta(x(0), H \cap C(x')) = \delta(x(0), \{[x, y] : x, y \in T'\})$. Indeed, if $[a, b]$ is any line segment from $H \cap C(x')$ then there are points $x, y \in T'$ such that line segments $[a, b]$ and $[x, y]$ are intersected by the same hyperplane and therefore $\delta(x(0), [x, y]) = \delta(x(0), [a, b])$. More precisely, for each hyperplane $x_j = x(0)_j$, $j = 1, \ldots, n$, which intersects $H \cap C(x')$, the set $\{w^k : k = 0, \ldots, m+1\}$ includes points which are from different sides. For a given line segment we can easily determine another, with endpoints from $\{w^k : k = 0, \ldots, m+1\}$ such that both are intersected by the same hyperplane. If one or both endpoints of $[a, b]$ are in some hyperplane $x_j = x(0)_j$, then whether or not we consider these points as an intersection of $[a, b]$ and hyperplane $x_j = x(0)_j$, we will be still able to find points x, y such that line segments $[a, b]$ and $[x, y]$ are intersected by the same hyperplane. □

The previous theorems enable us to organize directional rounding of $Plane(X(\overline{B}))$ in the following way. Choose any extreme point x' of $Plane(X(\overline{B}))$

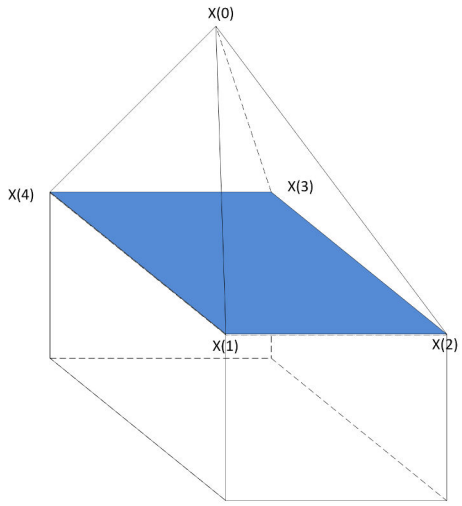

Figure 4.5: Polyhedron of LP relaxation

and identify plane $conv(Y)$, where $Y = \{y'(h) : h \in \overline{B}\}$ denotes the set of points $y'(h)$ such that directional rounding of all points from open sub–edge $]x', y'(h)[$ yield the same 0–1 point. Then perform directional rounding of the truncated cone $C(x')$ using theorem 4.4.2 and repeat the overall procedure on $Plane(X(\overline{B})) - C(x')$. At each iteration this procedure reduces the size of a set to be examined in the next iteration, in a fashion that assures the procedure is convergent.

4.4.3 Illustration of convergence of Scatter Search

In this subsection, we give two examples to illustrate the execution of the scatter search based on directional rounding. The first example with $n = 3$ is provided in order to illustrate graphically the process of the Convergent Scatter Search. The next example with $n = 6$ used to describe the execution of the scatter search algorithm step by step.

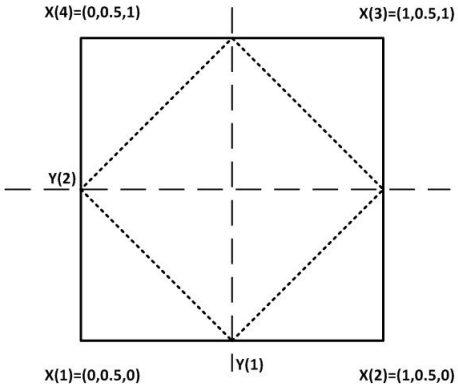

Figure 4.6: Plane $X(\overline{B})$ before starting CSS- dotted line represent cones that will be cut

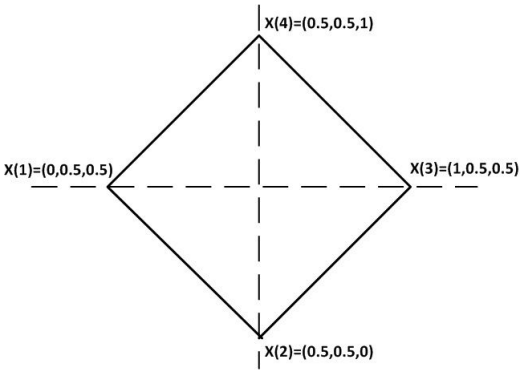

Figure 4.7: Plane $X(\overline{B})$ after cutting 4 cones

Example 4 *Consider the following 0–1 mixed integer problem*

$$\begin{array}{ll} \max & z = 1200x_1 + 2400x_2 + 500x_3 \\ s.t. & 2x_2 + 2x_3 \leq 3 \\ & 2x_1 + 2x_2 \leq 3 \\ & -2x_1 + 2x_2 \leq 1 \\ & 2x_2 - 2x_3 \leq 1 \\ & x_1, x_2, x_3 \in \{0, 1\}. \end{array} \tag{4.25}$$

The LP relaxation polyhedron of this problem is shown in Figure 4.5. The optimal solution of the LP relaxation is the point $x(0) = (0.5, 1, 0.5)$ while its adjacent extreme points are $x(1) = (0, 0.5, 0)$, $x(2) = (1, 0.5, 0)$, $x(3) = (0, 0.5, 1)$ and $x(4) = (1, 0.5, 1)$. The remaining extreme points are $(0, 0, 0)$, $(1, 0, 0)$, $(1, 1, 0)$ and $(0, 0, 1)$. The region $Plane(X(\overline{B}))$ determined by the points $x(1), x(2), x(3)$ and $x(4)$ is colored blue in Figure 4.5. Now, our procedure works as follows. First we choose any extreme point x' of $Plane(X(\overline{B}))$, say $x' = x(1)$. After that on each of edges $[x(1), x(2)]$ and $[x(1), x(4)]$ we identify points $y(1)$ and $y(2)$ closest to the $x(1)$ such that $y(1)_j = x(0)_j$ and $y(2)_k = x(0)_k$ for some j and k. In the particular case the point $y(1)$ is obtained as the intersection of edge $[x(1), x(2)]$ and hyperplane $x_1 = 0.5$, while the point y_2 is the intersection point of edge $[x(1), x(3)]$ and hyperplane $x_3 = 0.5$ (see Figure 4.6). The cone $C(x')$ to be directionally rounded is the

triangle with vertices $x', y(1), y(2)$. Its directional rounding produces just one 0-1 point $(0,0,0)$. In a similar way, we determine and directionally round cones originating at $x(2), x(3), x(4)$ to obtain points $(1,0,0), (0,0,1), (1,0,1)$. After eliminating all cones that we directionally rounded, i.e., all parts of $Plane(X(\overline{B}))$ that we have explored, we obtain the plane presented in Figure 4.7 with extreme points $x(1) = (0, 0.5, 0.5)$, $x(2) = (0.5, 0.5, 0)$, $x(3) = (1, 0.5, 0.5)$ and $x(4) = (0.5, 0.5, 1)$. Now we choose the point $x(1)$ as point x'. Its corresponding points y coincide with points $x(2)$ and $x(4)$ and therefore cone $C(x')$ is the triangle generated by the vertices $x(1)$, $x(2)$ and $x(4)$. The directional rounding of the cone $C(x')$ yields two points $(0,0,0)$ and $(0,0,1)$. Similarly, the directional rounding of the non-examined cone $C(x(3))$ produces two points $(1,0,0)$ and $(1,0,1)$. This completes our exploration. The procedure reports solution $(1,0,1)$ as optimal. Because of the simplicity of this problem we did not indicate how we actually performed the directional rounding of a cone step by step.

In the next example we show how the second variant of Convergent Scatter Search works.

Example 5

$$
\begin{aligned}
\max \quad & z = 100x_1 + 600x_2 + 1200x_3 + 2400x_4 + 500x_5 + 2000x_6 \\
s.t. \quad & \\
& 8x_1 + 12x_2 + 13x_3 + 64x_4 + 22x_5 + 41x_6 \leq 80 \\
& 8x_1 + 12x_2 + 13x_3 + 75x_4 + 22x_5 + 41x_6 \leq 96 \\
& 3x_1 + 6x_2 + 4x_3 + 18x_4 + 6x_5 + 4x_6 \leq 20 \\
& 5x_1 + 10x_2 + 8x_3 + 32x_4 + 6x_5 + 12x_6 \leq 36 \\
& 5x_1 + 13x_2 + 8x_3 + 42x_4 + 6x_5 + 20x_6 \leq 44 \\
& 5x_1 + 13x_2 + 8x_3 + 48x_4 + 6x_5 + 20x_6 \leq 48 \\
& 8x_5 \leq 10 \\
& 3x_1 + 4x_3 + 8x_5 \leq 18 \\
& 3x_1 + 2x_2 + 4x_3 + 8x_5 + 4x_6 \leq 22 \\
& 3x_1 + 2x_2 + 4x_3 + 8x_4 + 8x_5 + 4x_6 \leq 24 \\
& x_1, x_2, x_3, x_4, x_5, x_6 \in \{0,1\}
\end{aligned}
\tag{4.26}
$$

The optimal solution of the LP relaxation is the point $x(0) = (0, 0, 1, 0.36, 0.12, 1)$ with objective function value 4134.74. The extreme points adjacent to $x(0)$

are the following 6 points:
$x(1) = (1, 0, 1, 0.25, 0.10, 1)$
$x(2) = (0, 0.92, 1, 0, 0.68, 1)$
$x(3) = (0, 0, 0, 0.54, 0.19, 1)$
$x(4) = (0, 0, 1, 0.59, 0.65, 0.36)$
$x(5) = (0, 0, 1, 0.38, 0, 1)$
$x(6) = (0, 0, 1, 0.06, 1, 1)$.
These points are contained in the hyperplane P given by the equation $0.1x_1 + 0.18x_2 + 0.22x_3 + x_4 + 0.32x_5 + 0.65x_6 = 1.26$. The optimal vertex in P w.r.t to the objective function is the point $x(5)$ while remaining extreme points are adjacent to it. The $y(h)$ points on each edge $[x(5), x(h)]$ for $h \in \{1, 2, 3, 4, 6\}$ are the following 5 points:
$y(1) = (0.13, 0, 1, 0.36, 0.01, 1)$
$y(2) = (0, 0.04, 1, 0.36, 0.03, 1)$
$y(3) = (0, 0, 0.33, 0.49, 0.12, 1)$
$y(4) = (0, 0, 1, 0.42, 0.12, 0.88)$
$y(5) = (0, 0, 1, 0.36, 0.06, 1)$
while the best solution obtained by directionally rounding the cone originating at $x(5)$ and bounded by the face which contains points $y(h)$ is the point $(0, 0, 0, 1, 0, 0)$ with objective function value of 2400. The hyperplane H containing points $y(h)$ and $x(0)$ is defined as $0.19x_3 + x_4 + 0.480x_6 = 1.03$.

In the second iteration we repeat the overall procedure on the new problem derived from the staring problem by imposing two cuts, one defined by hyperplane P which excludes the point $x(0)$ and another defined by hyperplane H which excludes the point $x(5)$. The optimal solution of the new problem is the point $x(0) = (0, 0.04, 1, 0.36, 0.03, 1)$ with objective function value 4113.17 and its adjacent extreme points are:
$x(1) = (0.14, 0, 1, 0.36, 0.01, 1)$
$x(2) = (0, 0, 0.33, 0.49, 0.13, 1)$
$x(3) = (0, 0, 1, 0.42, 0.13, 0.88)$
$x(4) = (0, 0, 1, 0.36, 0.06, 1)$
$x(5) = (0, 0.06, 1, 0.36, 0, 1)$
$x(6) = (0, 0.91, 1, 0, 0.68, 1)$
while the hyperplane P that contains them is defined as $0.077x_1 + 0.23x_2 +$

$0.21x_3 + x_4 + 0.24x_5 + 0.61x_6 = 1.20$. The optimal vertex in P w.r.t to the objective function is the point $x(5)$ while the remaining extreme points are adjacent to it. The $y(h)$ points on each edge $[x(5), x(h)]$ for $h \in \{1, 2, 3, 4, 6\}$ are:
$y(1) = (0, 0.04, 0.83, 0.39, 0.03, 1)$
$y(2) = (0, 0.04, 1, 0.38, 0.03, 0.97)$
$y(3) = (0, 0.04, 1, 0.36, 0.01, 1)$
$y(4) = (0, 0.10, 1, 0.35, 0.03, 1)$
$y(5) = (0.03, 0.04, 1, 0.36, 0.00, 1)$.
The best solution obtained by directionally rounding the cone $C(x(5))$ is the point $(0, 1, 1, 0, 0, 1)$ with objective function value of 3800 which corresponds to the optimal solution of the original problem. The hyperplane H which contains the $y(h)$ points is given as $0.31x_2 + 0.19x_3 + x_4 + 0.48x_6 = 1.04$. The overall procedure is then repeated on the new problem adding two cuts as in the previous iteration. However, it should be emphasized that this small example discloses our convergence method can consume a long time to establish optimality in spite of finding an optimal solution on the second iteration. Therefore, we do not illustrate all iterations required to prove optimality.

4.5 One pass scatter search with directional rounding

According to the theorems of preceding sections, the search for an optimal MIP solution x^{opt} may be converted into finding a convex set $X \subset Face(X(\overline{B}))$ such that $x^{opt} \in \delta(x(0), X)$ where $x(0)$ is the extreme LP solution associated with the basis B. Because of the convexity of X it is natural to use the Scatter Search approach of combining two or more solutions in order to enhance the quality of current solutions. In our case, we restrict such solution combinations to convex combinations.

One method for solving MIP that results by combining Scatter search and

directional rounding may be outlined as follows:

Algorithm 47: Basic Scatter Search with Directional Rounding $BSSDR()$

Function `BSSDR`()

1 Step 1. Choose an LP basis solution $x(0)$;

2 Step 2. Determine an hyperplane H that excludes $x(0)$;

3 Step 3. Select a reference set $X \subseteq H \cap \overline{X}$. Then perform directional rounding by reference to the base point $x(0)$, creating points of the set $X^* = \delta(x(0), X)$;

4 Step 4. Return $x^* = argmin\{cx : x \in X^*, x \text{ MIP feasible}\}$ as a best solution to MIP.

Before implementing Basic (One-Pass) Scatter Search with directional rounding, several questions have to be answered, referring to the best way of choosing the extreme point $x(0)$, the valid cutting plane H and the subset X of points to be directionally rounded. In the following sections we undertake to answer these questions through an empirical study.

4.5.1 How to choose LP basis solution $x(0)$

In order to test the influence of the way in which the LP basis solution $x(0)$ is chosen we have conducted the following experiment. Firstly, we generate eleven instances of $x(0)$, consisting of an optimal solution of the LP–relaxation plus 10 additional LP basic solutions. Let $MaxIter$ equals the number of iterations performed by the Simplex algorithm of CPLEX to solve the LP–relaxation to optimality. Then we set the maximum number of simplex iterations to be performed to $MaxLimit_k$ where $k \in \{1, \ldots, 10\}$ and $MaxLimit_k$ chosen randomly between 1 and $MaxIter - 1$. In that way, the CPLEX LP solver returns 10 different points $x(0)$.

Next we generate the set X as a set of lines connecting the extreme points adjacent to $x(0)$. Finally, we round directionally each of these lines using the procedure described in Section 4.2.4 and report the best integer solution found. The experiment is performed on the first instance which belongs to the group of instances for multidimensional knapsack problem (MDKP) with $n = 500$ and $m = 5$, taken from OR-Library (see section 4.7). The results are

presented in Table 4.1. For each of LP basis solution we report the value of LP solution $x(0)^k$ returned by CPLEX (where $x(0)^0$ is the optimal solution of the LP–relaxation) and value of the best integer solution x^* found. Further, in column `Dev.`, we report the percentage deviation of the solution value of x^* from the best known value for the tested instance. It appears that better LP solutions yield better integer solutions. Therefore, we consider our best choice for $x(0)$ to be an optimal solution of the LP relaxation.

$x(0)^k$	$MaxLimit_k$	$cx(0)^k$	cx^*	Dev
$x(0)^0$	446	120234.92	119920	0.19
$x(0)^1$	41	34041.00	36177	69.89
$x(0)^2$	71	58185.00	60321	49.79
$x(0)^3$	149	106295.27	106377	11.46
$x(0)^4$	153	106656.75	106683	11.21
$x(0)^5$	166	108022.07	107998	10.11
$x(0)^6$	230	113361.49	113185	5.80
$x(0)^7$	252	114835.64	114724	4.51
$x(0)^8$	377	119498.12	119192	0.80
$x(0)^9$	387	119677.19	119403	0.62
$x(0)^{10}$	428	120160.07	119861	0.24

Table 4.1: Influence of chosen LP extreme point for MKP–5–500–1

4.5.2 How to choose the hyperplane H

Let B be the basis associated with $x(0)$. Then a valid cutting plane H may be defined as one spanned by points belonging to $X(\overline{B}) = \{x(j) = x(0) - \theta_j^* D_j, j \in \overline{B}\}$. Clearly, different values of θ_j^* generate different hyperplanes. Therefore, we provide several options for setting the values of θ_j^* and then test the influence of the chosen θ_j^* values on the solution process.

Hyperplane H^1 [93]. The first (basic) option which naturally arises from the previous discussion is to choose θ_j^* values to yield extreme points adjacent to $x(0)$. However, in the case of degeneracy some of these values may equal 0, and therefore extreme points corresponding to such θ_j^* values will be the same as $x(0)$ itself. In order to avoid duplication of the extreme point $x(0)$ and to generate a larger set of reference points we set such θ_j to a positive value,

which we have chosen to equal the smallest of the values of those for which θ_j^* that are positive. More precisely, if $\theta_j^* = 0$ set $\theta_j^* = \min\{\theta_k^* : \theta_k^* > 0, k \in \overline{B}\}$.

Hyperplane H^2 [93]. Let z', z^* and $z(0)$ respectively denote the optimum LP value, the value of the best known solution and the objective value of the extreme point $x(0)$. Further, let r_j denote the reduced cost associated with the non basic variable indexed by $j \in \overline{B}$. Now, for each point $x(j) = x(0) - \theta_j^* D_j$ we can calculate the value $z(j)$ of the objective function at $x(j)$ as:

$$z(j) = z(0) - \theta_j^* D_{jj} r_j, \quad j \in \overline{B}, \tag{4.27}$$

where D_{jj} represents the value of j^{th} entry in the vector D_j, which corresponds to the nonbasic variable x_j and which is 1 or -1 depending on whether the non-basic variable is at its upper or lower bound. The value of θ_j^* is deduced from (4.27) by setting $z(j)$ equal to one of the predefined values z^+ or z^- depending on whether $D_{jj} r_j < 0$ or not. The values of z^+ and z^- are defined according to whether $z^* \geq z(0)$, as follows:

- if $z^* \geq z(0)$: $z^+ = z^* + \beta(z' - z(0))$ and $z^- = z(0) - \beta(z' - z(0))$
- if $z^* < z(0)$: $z^+ = z(0) + \beta(z' - z(0))$ and $z^- = z^* - \beta(z' - z(0))$

where β is a scalar parameter in the interval $]0, 1[$. In our implementation we set $\beta = 0.3$.

Hyperplane H^3 [93]. The hyperplane H^3 is obtained from the option to generate the hyperplane H^2 by repairing the feasibility of each infeasible point produced. The feasibility of each infeasible point $x(j)$ is repaired by setting θ_j to the greatest value such that the lower and upper bounds of all variables are satisfied. In other words, feasibility is repaired by moving each infeasible point to the closet extreme point that belongs to the line segment $[x(0), x(j)]$.

In order to test the performances of these three options we implemented three variants of One Pass Scatter Search with Directional Rounding. Each of these variants has been developed according to the steps presented in section 4.5, using different ways of determining θ_j^* values (and therefore determining

different $X(\bar{B})$ sets). According to the option which is used for determining θ_j^* values, these three variants are named SP_H^1, SP_H^2 and SP_H^3. Each algorithm SP_H^k starts by solving the LP relaxation problem to obtain $x(0)$ and then determines the set $X(\bar{B})$ relative to the optimal solution of the LP relaxation and the chosen option for determining θ_j^*. Then directional rounding is performed on each point belonging to the line segment $[x, y]$, $x, y \in X(\bar{B})$ as well as on each point from the line segment $[x, z]$ where $x \in X(\bar{B})$ and z is the center of gravity of the remaining points in $X(\bar{B})$ (different from x). In order to do better, in the case where one of options 1 or 3 is used for determining θ_j^* values, each line segment $[x, z]$ is extended to the boundary of the feasible region, i.e., $[x, z] \subset [x, t]$, t–boundary point, and after that each point from the line segment $[x, t]$ is directionally rounded. In the case when option 2 is used for determining θ_j^* values, such an extension is not performed because it might happen that both points, x and z, are infeasible and therefore $Line(x, z)$ will not intersect the feasible region. The reason why we include line segments which passes through the point z is provided in Corollary 4.2.10. This theorem states that the directional rounding of two line segments, whose endpoints belong to the same rays starting at $x(0)$, yields the same set of 0–1 solutions.

The numerical experiments have been carried out on 30 instances for multidimensional knapsack problem with $n = 500$ and $m = 5$ (see section 4.7). The obtained results are presented in Table 4.2. For each variant, we report the value of the best solution found (Column cx^*), the average value of all feasible solutions met during its execution (Column `Avg`) and the CPU time in seconds which corresponds to the total execution CPU time of the procedure (Column `Time`). Regarding the overall average value of all solutions encountered it appears that the procedure SP_H^3 is best. However, SP_H^2 is the fastest method in terms of the average CPU time consumed. Our results show that all variants offer the same solution as the best and hence it follows that directional rounding of some boundary point of each hyperplane yields that solution.

instance	StarPath						
	cx^*	SP_H^1		SP_H^2		SP_H^3	
		Avg.	Time	Avg.	Time	Avg.	Time
1	119920	118913.8	32.2	118928.0	28.1	118913.4	32.5
2	117652	115977.3	34.1	115975.4	30.1	115977.3	34.4
3	120958	118815.2	34.8	118958.1	31.0	118811.0	35.2
4	120695	119268.8	32.8	119246.1	29.0	119269.6	33.1
5	122061	120658.1	32.8	120655.9	28.9	120657.7	33.1
6	121776	120401.2	33.2	120400.8	29.1	120401.2	33.4
7	118997	117884.5	32.2	117884.5	28.3	117884.5	32.6
8	120335	119128.4	32.4	119119.8	28.3	119128.4	32.6
9	121287	119854.8	33.1	119855.9	29.2	119854.8	33.3
10	120509	119012.9	32.9	119029.3	28.8	119012.5	33.1
11	218252	176805.0	34.5	202104.4	30.0	176795.2	34.7
12	221027	187821.0	35.6	218816.2	30.8	187828.8	35.9
13	217304	170189.7	34.2	161414.3	29.9	170184.2	34.8
14	223343	192930.8	33.7	213995.7	29.3	192960.6	34.0
15	218806	179357.0	33.9	208388.1	28.9	179426.5	34.0
16	220324	170050.0	34.3	164525.2	28.4	170008.7	33.3
17	219914	179786.8	34.2	174866.0	29.9	179756.2	34.5
18	218041	187873.7	34.1	178300.4	29.9	187859.4	34.3
19	216790	185429.3	33.8	173864.7	29.4	185416.0	34.1
20	219652	197396.0	34.4	197543.2	30.1	197406.9	34.5
21	295745	180607.4	33.8	160295.9	29.1	180703.2	34.2
22	307962	198753.6	33.7	163204.8	28.8	198695.5	34.7
23	299694	192086.7	34.6	166511.5	29.7	191988.0	34.9
24	306335	188726.3	34.0	166301.6	29.0	188705.2	34.3
25	300265	192278.0	35.2	159237.1	30.5	192425.1	35.5
26	302433	183770.0	33.4	160240.5	28.1	183720.1	33.6
27	301193	184915.8	35.6	157318.4	30.4	184890.4	35.8
28	306286	186903.8	34.4	163453.9	29.5	186989.7	34.7
29	302696	184193.2	33.7	154059.9	28.6	184143.6	33.8
30	299859	191705.7	33.9	159687.4	28.8	191975.8	34.0
Avg.	214003.70	163383.16	33.84	156472.76	29.33	163392.99	34.10

Table 4.2: Computational results on 30 instances of MKP with $n = 500$ and $m = 5$

4.5.3 How to choose the reference set X

Similarly, just as there are several ways to determine the set $X(\overline{B})$ there are also several ways to choose the reference set X, which is to be directionally rounded. As already shown, we have an efficient procedure for rounding lines and consequently we will only consider the reference set X as a set of lines belonging to $Face(X(\overline{B}))$ that will be directionally rounded. Therefore, we characterize set X as $X = \{[x', x''] : x', x'' \in \widetilde{X}\}$ where set $\widetilde{X}$ represents a set of points from $Face(X(\overline{B}))$. Now, it is obvious that the set X is directly correlated with set $\widetilde{X}$. Thus since $\widetilde{X}$ may be determined in several different ways, we also have several different sets X of lines to be directionally rounded.

Reference set X^1 [93]. The first and natural choice is given by $\widetilde{X} = \widetilde{X}^1$ as the set of extreme points of $Face(X(\overline{B}))$. In that case, if $Face(X(\overline{B}))$ has n extreme points, the set denoted by X^1 will contain $O(n^2)$ lines and the endpoints of these lines will be the extreme points of $Face(X(\overline{B}))$.

Reference set X^2**.** The reference set $\widetilde{X}^1$ can be extended by adding points that belong to the extreme rays of $Face(X(\overline{B}))$ and which have at least on coordinate equal to the corresponding coordinate of $x(0)$. More precisely, for each pair $x', x'' \in \widetilde{X}^1$ the points to be added from each segment $[x', x'']$ are determined as points $x(\lambda) = x' + \lambda(x'' - x')$ such that $x_j = x_j^*$ for some $j \in N$. From Lemma 4.2.5 follows that the λ values that correspond to points to be added are actually λ_j^* values given as $\lambda_j^* = \frac{x_j^* - x_j'}{x_j'' - x_j'}$ for $j \in N^{\neq}(x', x'') = \{j \in N : x_j'' \neq x_j'\}$ which belongs to the interval $]0, 1[$. Hence, the extended set $\widetilde{X}^2 = \widetilde{X}^1 \cup \{x' + \lambda_j^*(x'' - x') : x', x'' \in \widetilde{X}^1, j \in N^{\neq}(x', x'')\}$ and the set X will be denoted by $X^2 = \{[x, y] : x, y \in \widetilde{X}^2\}$. The motivation for extending set $\widetilde{X}^1$ relies not only on the fact that a larger set $\widetilde{X}^1$ affords a higher probability of achieving good solutions but also on the fact that if the set $\widetilde{X}^1$ is extended as previously described, the interior of $Face(X(\overline{B}))$ will be explored more effectively. For example, if $Face(X(\overline{B}))$ is a simplex then without extending set $\widetilde{X}^1$ we will explore just the boundary of $Face(X(\overline{B}))$. Also, the decision to extend set $\widetilde{X}^1$ in this way is motivated by the fact that the added points represent switching points on each boundary edge, as was shown in Theorem 4.2.1. Further, it should be emphasized that if the set X is determined relative to the set $\widetilde{X}^2$, the one pass heuristic resembles the procedure used for rounding a cone within the Convergent Scatter Search algorithm.

Reference set X^3**.** Rounding all lines with endpoints from $\widetilde{X}^2$ may be highly time consuming since the number of lines to be rounded is $O(n^6)$ in the worst case. However, the number of lines to be rounded may be reduced drastically if we omit rounding the lines for which at least one endpoint is not extreme point of $Face(X(\overline{B}))$. The resulting set of lines is denoted by X^3, and the total number of lines to be rounded becomes $O(n^4)$.

As, suggested in Glover [93] both sets $\widetilde{X}^k$ could be augmented by adding to them the point that corresponds to the center of gravity of points from the set $\widetilde{X}^k$ as well as all centers of gravity of subsets consisting of $|\widetilde{X}^k| - 1$ points

from set $\widetilde{X}^k$.

In order to test these three possibilities for generating set X we implemented three One Pass Scatter Search with Directional Rounding heuristics following steps of Algorithm 47. The ways of choosing a point $x(0)$ and a hyperplane H are the same for all of them. The LP basis solution $x(0)$ is chosen as an optimal solution of LP relaxation, while hyperplane H corresponds to the hyperplane spanned by the extreme points adjacent to $x(0)$. The only difference among them is the set X which is directionally rounded. According to the set which is used as X, those three variants are referred as: SP_X^1, SP_X^2 and SP_X^3. The numerical experiments have been, again, performed on 30 instances for multidimensional knapsack problems with $n = 500$ and $m = 5$ (see section 4.7). Because of the CPU time consumed by one pass heuristic when X is equal to either X^2 or X^3, we have imposed the time limit of 100 seconds in the case $X = X^2$ and 60 seconds in the case $X = X^3$. However, in order to achieve better integer solutions within an imposed time limit we sort the extreme points adjacent to $x(0)$ in descending order according to their objective values. Therefore, during the directionally rounding process we give priority to lines whose endpoints have greater objective values. The results obtained during these experiments are presented in Table 4.3. For each tested variant we report the best integer solution value found (Column cx^*), the time when it is found (Column `time`) as well as the total consumed time (Column `total time`). As expected, heuristics SP_X^2 and SP_X^3 are slower then the heuristic SP_X^1. However, on average, heuristic SP_X^2 offers slightly worse solutions than the other two variants. The reason is that the variants that use X^2 and X^3 directionally round a greater number of lines than the variant using X^1 and thus consume a greater amount of CPU time. On the other hand, the greater number of lines that are rounded, the greater the probability of finding a good solution. This fact explains why the variant which uses X^2 is capable of finding better solutions than the others.

ins.	SP_X^1			SP_X^2			SP_X^3		
no	cx^*	time	total time	cx^*	time	total time	cx^*	time	total time
1	119920	12.2	13.1	119968	91.5	100.4	119936	59.6	60.0
2	117652	0.1	23.3	117746	10.4	100.7	117725	3.5	60.0
3	120958	0.1	24.5	120998	43.1	100.4	120962	25.0	60.0
4	120695	14.3	17.8	120695	26.2	100.4	120695	0.1	60.0
5	122061	11.2	16.4	122181	61.6	100.3	122163	36.3	60.0
6	121776	2.5	18.1	121910	17.6	100.4	121869	0.0	60.0
7	118997	3.1	13.4	119021	90.4	100.4	119021	50.3	60.0
8	120335	11.6	14.1	120465	2.2	100.4	120467	30.5	60.0
9	121287	16.7	18.9	121364	11.0	100.5	121371	57.3	60.0
10	120509	13.1	16.5	120579	73.0	100.4	120621	57.1	60.0
11	218252	20.0	22.1	218292	71.8	100.3	218271	57.7	60.0
12	221027	19.7	23.9	221085	24.9	100.6	221038	33.5	60.0
13	217304	0.0	19.3	217436	43.2	100.4	217364	59.4	60.0
14	223343	0.1	18.2	223450	90.8	100.6	223450	29.4	60.0
15	218806	0.1	15.9	218901	41.6	100.2	218862	49.7	60.0
16	220324	6.4	13.4	220421	5.6	100.4	220376	0.2	60.0
17	219914	21.3	21.6	219914	9.7	100.4	219918	18.3	60.0
18	218041	16.1	18.4	218132	10.8	100.3	218100	30.8	60.0
19	216790	8.9	17.7	216907	5.9	100.5	216836	30.5	60.0
20	219652	13.9	18.3	219667	58.3	100.3	219590	0.0	60.0
21	295745	9.6	13.4	295745	1.8	100.2	295745	12.6	60.0
22	307962	7.5	14.7	307974	85.3	100.2	307962	2.7	60.0
23	299694	0.2	19.2	299740	17.3	100.5	299694	2.5	60.0
24	306335	5.5	16.6	306377	24.8	100.4	306335	1.5	60.0
25	300265	9.7	21.0	300268	62.5	100.5	300226	5.7	60.0
26	302433	0.0	13.2	302517	31.6	100.2	302439	28.4	60.0
27	301193	0.1	21.1	301262	54.6	100.7	301233	58.4	60.0
28	306286	7.2	15.6	306349	87.2	100.3	306336	18.7	60.0
29	302696	9.7	12.6	302736	71.7	100.2	302696	3.9	60.0
30	299859	0.0	12.7	299859	1.6	100.4	299859	0.0	60.0
Avg.	214003.70	8.03	17.49	214065.30	40.93	100.40	214038.67	25.46	60.00

Table 4.3: Testing different options for choosing set X

4.6 Convergent scatter search with directional rounding as a heuristic

In this section we provide one way in which the Convergent Scatter search with directional rounding presented in Section 4.4 may be used as a heuristic. Instead of repetitively exploiting the entire hyperplane, one can stop execution of the algorithm after a predefined number of iterations. In that way, the optimality of the obtained solution can not be guaranteed but we are motivated to investigate whether such a solution may nevertheless be of high quality. The heuristic we propose is quite simple. It starts choosing an extreme point of a LP polyhedron, determines a hyperplane spanned by the extreme points adjacent to the previously chosen extreme point and directionally rounds

each corner of the hyperplane thus determined. The best solution found is reported as a solution of the input problem. However, depending on problem complexity, sometimes even directional rounding of one corner may be so time expensive and therefore some time limit on directional rounding of a corner must be imposed. Also, directional rounding of all corners sometimes is time wasting, and in this case it can be better to round just some small number of corners, selecting the k best corners to be directionally rounded as those originating at the k best extreme points of a given hyperplane. Note that such an approach can be easily parallelized, rounding several corners at once.

We have tested the performance of this approach on small as well as large scale multidimensional knapsack problems. For small problems each corner is rounded without any time limit, while for large scale problems 100 corners are rounded with the time limit for rounding a corner set to 1 second. The description of small problems is presented in Table 4.4. For each small problem values of n and m are provided as well as the optimum value. These problems have been taken from the OR-library. The large scale problems are same as those used in previous tests and consist of 30 instances for MDKP with $n = 500$ and $m = 5$. The results obtained are compared against those obtained on same problems by three different options, previously described, for choosing set X in the hyperplane spanned by extreme points adjacent to an optimal solution of the LP relaxation. For each heuristic we report the best solution value found (Column cx^*) as well as the time taken to find that solution (Column `time`).

The results obtained on small instances (Table 4.4) give the following ranking of the heuristics according to the number of optimal solutions found. Heuristic SP_X^2 is the best since it succeeds in solving 6 out of 7 instances to optimality. The second place heuristic derives from convergent scatter search ($HCSS$) and the SP_X^3 heuristic, both of which succeeded in solving 4 instances to optimality. Finally, the last place heuristic turned to be SP_X^1 heuristic which succeeded in solving just 3 instances to optimality.

Comparing the average results obtained on larger instances (Table 4.5) yields the following ranking. The best performing heuristic again is SP_X^2, the second best is the heuristic derived from convergent scatter search, while the third best is the SP_X^3 heuristic. Finally, the last place again is taken

by SP_X^1. Regarding average computational time the SP_X^1 heuristic significantly outperforms all the others. The second fastest option is the SP_X^3 heuristic, while heuristics SP_X^2 and $HCSS$ are the slowest.

no	n	m	optimal	SP_X^1	SP_X^2	SP_X^3	HCSS
1	6	10	3800	3800	3800	3800	3700
2	10	10	8706.1	8650.1	8706.1	8687.5	8706.1
3	15	10	4015	4015	4015	4015	4015
4	20	10	6120	6120	6120	6120	6110
5	28	10	12400	12380	12400	12380	12400
6	39	5	10618	10587	10618	10618	10618
7	50	5	16537	16440	16504	16499	16499

Table 4.4: Comparison on small problems

no	SP_X^1		SP_X^2		SP_X^3		HCSS	
instance	cx^*	time	cx^*	time	cx^*	time	cx^*	time
1	119920	12.2	119968	91.5	119936	59.6	119968	37.3
2	117652	0.1	117746	10.4	117725	3.5	117746	3.5
3	120958	0.1	120998	43.1	120962	25.0	120998	9.6
4	120695	14.3	120695	26.2	120695	0.1	120591	79.8
5	122061	11.2	122181	61.6	122163	36.3	122157	9.8
6	121776	2.5	121910	17.6	121869	0.0	121918	65.1
7	118997	3.1	119021	90.4	119021	50.3	119021	38.1
8	120335	11.6	120465	2.2	120467	30.5	120457	96.6
9	121287	16.7	121364	11.0	121371	57.3	121507	63.3
10	120509	13.1	120579	73.0	120621	57.1	120555	91.8
11	218252	20.0	218292	71.8	218271	57.7	218261	76.4
12	221027	19.7	221085	24.9	221038	33.5	221085	7.8
13	217304	0.0	217436	43.2	217364	59.4	217429	90.5
14	223343	0.1	223450	90.8	223450	29.4	223450	28.6
15	218806	0.1	218901	41.6	218862	49.7	218884	21.5
16	220324	6.4	220421	5.6	220376	0.2	220394	10.0
17	219914	21.3	219914	9.7	219918	18.3	219786	37.9
18	218041	16.1	218132	10.8	218100	30.8	218094	76.4
19	216790	8.9	216907	5.9	216836	30.5	216839	1.4
20	219652	13.9	219667	58.3	219590	0.0	219624	1.1
21	295745	9.6	295745	1.8	295745	12.6	295696	19.6
22	307962	7.5	307974	85.3	307962	2.7	308019	23.6
23	299694	0.2	299740	17.3	299694	2.5	299714	6.8
24	306335	5.5	306377	24.8	306335	1.5	306369	17.3
25	300265	9.7	300268	62.5	300226	5.7	300262	74.5
26	302433	0.0	302517	31.6	302439	28.4	302467	22.5
27	301193	0.1	301262	54.6	301233	58.4	301246	64.1
28	306286	7.2	306349	87.2	306336	18.7	306349	100.5
29	302696	9.7	302736	71.7	302696	3.9	302736	79.5
30	299859	0.0	299859	1.6	299859	0.0	299859	23.4
Avg.	214003.70	8.03	214065.30	40.93	214038.67	25.46	214049.37	42.61

Table 4.5: Comparison on large scale problems

4.7 Comparison of solutions returned by heuristics and best known solutions

The multidimensional knapsack problem (MKP) is a special case of a 0–1 mixed integer program and is known to be NP–hard. This problem has been widely studied in the literature, and efficient exact and approximate methods have been developed for obtaining optimal or near–optimal solutions, including very effective methods by Hanafi and Fréville (1998) [110], Vasquez and Hao (2001) [216], Vasquez and Vimont (2005) [217], Hanafi and Wilbaut (2008) [111], Boussier et al. (2010) [22], Khemakhem et al. (2012) [143]. Most of the best-known solutions for the set of MDKP instances available in the OR-Library [17] were obtained by Vasquez and Hao (2001), Vasquez and Vimont (2005) and Hanafi and Wilbaut (2008). A comprehensive annotated bibliography of exact and approximate algorithms for (MKP) appears in Fréville and Hanafi (2005) [79] and Wilbaut et al. (2008) [222].

In Table 4.6, we compare the quality of solutions obtained by three variants of One Pass scatter search heuristic with the directional rounding heuristics SP_X^1, SP_X^2, SP_X^3 and $HCSS$. For testing purposes, we used 30 instances of the multidimensional knapsack problem with $n = 500$ and $m = 5$. The solution quality returned by each heuristic is measured as the percentage deviation from the corresponding best-known solution values and these results are presented in Columns `Dev`. The results show that all heuristics succeed in finding high quality solutions. The average gap between the solution value returned by several heuristics and best known solution value is not greater than 0.10%. Moreover, every heuristic attains this level of performance for at least 20 instances. Regarding the deviations achieved over solving all instance, heuristic SP_X^1 has a maximum deviation of 0.24%, SP_X^2 has a maximum deviation of 0.17%, while heuristics SP_X^3 and $HCSS$ have maximum deviations of 0.18%.

no	best	SP_X^1		SP_X^2		SP_X^3		HCSS	
instance	known	cx^*	dev(%)	cx^*	dev(%)	cx^*	dev(%)	cx^*	dev(%)
1	120148	119920	0.19	119968	0.15	119936	0.18	119968	0.15
2	117879	117652	0.19	117746	0.11	117725	0.13	117746	0.11
3	121131	120958	0.14	120998	0.11	120962	0.14	120998	0.11
4	120804	120695	0.09	120695	0.09	120695	0.09	120591	0.18
5	122319	122061	0.21	122181	0.11	122163	0.13	122157	0.13
6	122024	121776	0.20	121910	0.09	121869	0.13	121918	0.09
7	119127	118997	0.11	119021	0.09	119021	0.09	119021	0.09
8	120568	120335	0.19	120465	0.09	120467	0.08	120457	0.09
9	121575	121287	0.24	121364	0.17	121371	0.17	121507	0.06
10	120717	120509	0.17	120579	0.11	120621	0.08	120555	0.13
11	218428	218252	0.08	218292	0.06	218271	0.07	218261	0.08
12	221202	221027	0.08	221085	0.05	221038	0.07	221085	0.05
13	217542	217304	0.11	217436	0.05	217364	0.08	217429	0.05
14	223560	223343	0.10	223450	0.05	223450	0.05	223450	0.05
15	218966	218806	0.07	218901	0.03	218862	0.05	218884	0.04
16	220530	220324	0.09	220421	0.05	220376	0.07	220394	0.06
17	219989	219914	0.03	219914	0.03	219918	0.03	219786	0.09
18	218215	218041	0.08	218132	0.04	218100	0.05	218094	0.06
19	216976	216790	0.09	216907	0.03	216836	0.06	216839	0.06
20	219719	219652	0.03	219667	0.02	219590	0.06	219624	0.04
21	295828	295745	0.03	295745	0.03	295745	0.03	295696	0.04
22	308086	307962	0.04	307974	0.04	307962	0.04	308019	0.02
23	299796	299694	0.03	299740	0.02	299694	0.03	299714	0.03
24	306480	306335	0.05	306377	0.03	306335	0.05	306369	0.04
25	300342	300265	0.03	300268	0.02	300226	0.04	300262	0.03
26	302571	302433	0.05	302517	0.02	302439	0.04	302467	0.03
27	301339	301193	0.05	301262	0.03	301233	0.04	301246	0.03
28	306454	306286	0.05	306349	0.03	306336	0.04	306349	0.03
29	302828	302696	0.04	302736	0.03	302696	0.04	302736	0.03
30	299910	299859	0.02	299859	0.02	299859	0.02	299859	0.02
Avg.	214168.4	214003.7	0.10	214065.3	0.06	214038.7	0.07	214049.4	0.07

Table 4.6: Comparison with best known solutions

4.8 Concluding remarks

In this chapter, the Convergent Scatter Search with directional rounding for solving 0–1 MIP problems is introduced for the first time. The idea for using Scatter search as a method for exploring points on the imposed cutting plane is justified by proving theorems which show that the cutting plane contains a polyhedral region which produces all feasible 0–1 solutions by directional rounding. To carry out directional rounding relative to a line segment, an efficient method is used. The work of Convergent Scatter Search has been demonstrated on small examples. Additionally, two variants of the implementation of Convergent Scatter Search have been described.

Additionally, we propose several heuristics for solving 0–1 MIP which

combine Scatter search and directional rounding. Accompanying this, an exhaustive empirical study is performed in order to find the best choice rules for producing a One-Pass directional rounding heuristic. In addition, we propose a heuristic derived from the Convergent Scatter search procedure and compare it with the other heuristics. Our findings demonstrate the efficacy of these first stage methods, which makes them attractive for use in situations where very high quality solutions are sought with an efficient investment of computational effort.

All of proposed methods can be easily parallelized to execute several tasks simultaneously. For example, in heuristics derived from Convergent Scatter Search with directional rounding, multiple cones originating at different extreme points can be directionally rounded at once. Similarly, in One-Pass Scatter Search with directional rounding several lines can be directionally rounded in parallel. Future work will include the parallelization of the proposed methods, their testing on other 0–1 MIP problems, as well as their extension to heuristics for general MIP problems. More advanced forms of the Scatter Search with directional rounding methods, which place increased emphasis on the ability to find globally best solutions by investing more extensive effort, will be the topic of a sequel.

Chapter 5

Concluding remarks

In this thesis we presented several practical and theoretical contributions to Variable Neighborhood Search (VNS) and Scatter Search metahuerstics as well as matheuristics (approaches that combine ideas of mathematical programming and metaheuristics). The contributions regarding Variable neighborhood search, as presented in Chapter 2, include new VNS variants as well as several efficient VNS based heuristics for solving eight NP-hard problems arising in transportation, logistic, scheduling, power generation and clustering. Moreover, for each of considered problems the proposed VNS heuristics turned out to be the new state-of-the-art heuristic approaches. Such performances of VNS heuristics as well as a wide range of tackled problems undoubtedly indicate that VNS is very powerful tool for solving optimization problems.

The contributions on matheuristic approaches have been presented in Chapter 3. Namely, we present two efficient approaches for creating an initial solution for a MIP problem as well as several hybrid approaches for solving Multicommodity Fixed-Charge Network Design (MCND) problem. In particular, for finding initial feasible solutions of mixed integer programs we proposed two heuristics named *variable neighborhood diving* (*VN diving*) and *single neighborhood diving* (*SN diving*). The proposed heuristics are based on systematic hard variable fixing (i.e., diving) in order to generate smaller subproblems whose feasible solution (if one exists) is also feasible for the original problem. VN diving, follows the principles of variable neighborhood decomposition search and uses pseudo-cuts which. are added during the search process in order to prevent exploration of already visited search space areas. On the other hand, SN diving explores only one neighborhood in each iteration. However, it uses a sophisticated mechanism to avoid the already visited solutions.

For solving Multicommodity Fixed-Charge Network Design (MCND) problem we proposed several heuristics that combine ideas of solving (optimally or near optimally) a series of small sub-problems obtained from a series of linear programming relaxations and ideas of Slope Scaling heuristics within the search for an optimal solution of a given problem. In particular we show that ideas of a convergent algorithm based on the LP-relaxation and pseudo-cuts may be successfully used to guide Slope Scaling heuristic during the search for an optimal (or near-optimal) solution and vice-versa. The computational results, obtained on the benchmark instances from the literature demonstrate the efficiency of the proposed matheuristics for MCND.

Chapter 4 is devoted to the contributions on Scatter Search metaheuristic. Building on properties of directional rounding as a mapping from continuous to discrete (binary) space and those of the solution space of 0-1 MIP in Chapter 4 we propose Convergent Scatter Search with directional rounding for solving 0–1 MIP problems. The key properties that are exploited within Convergent Scatter Search with directional rounding algorithm are the fact that directional rounding of a line, as embodied in a Star Path, contains a finite number of distinct 0–1 points and existence of the cutting plane that contains a polyhedral region which produces all feasible 0–1 solutions by directional rounding. More precisely, these properties enable us to organize the search for an optimal solution of 0–1 MIP problems using Scatter Search in association with both cutting plane and extreme point solution approaches. The proposed convergent algorithm is accompanied by the proof of its finite convergence as well as by two variants of its implementation and examples that illustrate the running of the approach. Starting from this exact approach and taking into account established properties of the solution space of 0-1 MIP we propose several heuristics for solving 0–1 MIP which combine Scatter search and directional rounding. The proposed heuristics are actually first stage implementations which aim is to establish the power of the methods in a simplified form. Thus, an exhaustive empirical study, in order to find the best ingredients of a such one method, as well as a computational testing on a test bed of 0-1 MIP problems, in order to disclose the merit of the proposed approaches, have been performed. Our findings demonstrate the efficacy of these first stage methods, which makes them attractive for use in situations

where very high quality solutions are sought with an efficient investment of computational effort.

The work presented in this thesis give us several future work directions. For example, efficiency of VNS heuristics presented in Chapter 2 represents good starting point for developing new VNS based methods for problems similar to ones presented in that chapter. Also, some problems presented in that chapter may be easily extended in the natural way taking into account practical considerations. For example, traveling salesman problem with draft limits and attractive traveling salesman problem may be extended to the ones with time windows (adding time windows constraints to the nodes), minimum sum-of-squares clustering on networks may be extended to the constrained minimum sum-of-squares clustering on networks (taking in to account must-link and cannot-link constraints on the elements, or a hierarchy which must be preserved in the clusters obtained), periodic maintenance problem may be generalized for example allowing service of more than one machine in each time period etc. Further, the work presented in Chapter 4 shows how generic heuristics for mathematical programming can be turned into complete solution methods. Since, such results are relatively rare in the literature, the interesting research direction may be developing novel heuristic techniques based on adaptations of such algorithms. Additionally, parallelization of the methods (exact and heuristic) in Chapter 4, their extension to methods for general MIP problems, and development of their more advanced forms seem as promising research directions. The quality of results obtained by the matherutistics developed in Chapter 3 indicates that the development of matheuristics based on hard variable fixing for mixed integer non-linear programming may be promising research area. Further, the matheuristic proposed in Chapter 3 for solving MCND seems as a good starting point for developing new Iterative linear programming-based heuristics that include some ideas of existing exact and heuristic approaches either for MCND or the problems related to it.

Bibliography

[1] R. Aboudi, A. Hallefjord, R. Helming, and K. Jornsten. A note on the Pivot and Complement heuristic for 0–1 programming problems. *Operations Research Letters*, 8:21–23, 1989.

[2] T. Achterberg and T. Berthold. Improving the feasibility pump. *Discrete Optimization*, 4(1):77–86, 2007.

[3] D. Aloise, A. Deshpande, P. Hansen, and P. Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75:245–248, 2009.

[4] D. Aloise, P. Hansen, and L. Liberti. An improved column generation algorithm for minimum sum-of-squares clustering. *Mathematical Programming*, 131:195–220, 2012.

[5] E. Balas. The Intersection Cut–A New Cutting Plane for Integer Programming. *Operations Research*, 19:19–39, 1971.

[6] E. Balas and R. Jeroslow. Canonical cuts on the unit hypercube. *SIAM Journal on Applied Mathematics*, 23(1):61–69, 1972.

[7] E. Balas and F. Margot. Generalized intersection cuts and a new cut generating paradigm. *Mathematical Programming*, 137:19–35, 2013.

[8] E. Balas and C. Martin. Pivot and Complement-A Heuristic for 0–1 Programming. *Managment Science*, 26:86–96, 1980.

[9] E. Balas and E. Zemel. An algorithm for large zero-one knapsack problems. *Operations Research*, pages 1130–1154, 1980.

[10] E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Mathematical Programming*, 58: 295–324, 1993.

[11] E. Balas, S. Ceria, M. Dawande, F. Margot, and G. Pataki. OCTANE: A New Heuristic for Pure 0-1 Programs. *Operations Research*, 49: 207–225, 2001.

[12] E. Balas, S. Schmieta, and C. Wallace. Pivot and shift-a mixed integer programming heuristic. *Discrete Optimization*, 1:3–12, 2004.

[13] M. O. Ball. Heuristics based on mathematical programming. *Surveys in Operations Research and Management Science*, 16(1):21–38, 2011.

[14] M. Battarra, A. A. Pessoa, A. Subramanian, and E. Uchoa. Exact algorithms for the traveling salesman problem with draft limits. *European Journal of Operational Research*, 235(1):115–128, 2014.

[15] J. E. Beasley. A note on solving large p-median problems. *European Journal of Operational Research*, 21:270–273, 1985.

[16] J. E. Beasley. Lagrangean heuristics for location problems. *European Journal of Operational Research*, 65(3):383–399, 1993.

[17] J. Beasly. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

[18] L. Bertacco, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 4(1):63–76, 2007.

[19] T. Berthold. Primal heuristics for mixed integer programs. Master's thesis, Technische Universitat Berlin, 2006.

[20] T. Berthold. RENS-relaxation enforced neighborhood search. Technical report tr-07-28, ZIB, Berlin, 2007.

[21] P. Bonami, M. Kilinç, and J. Linderoth. Algorithms and software for convex mixed integer nonlinear programs. In *Mixed integer nonlinear programming*, pages 1–39. Springer, 2012.

[22] S. Boussier, M. Vasquez, Y. Vimont, S. Hanafi, and P. Michelon. A multi-level search strategy for the 0–1 Multidimensional Knapsack. *Discrete Applied Mathematics*, 158:97–109, 2010.

[23] J. Brimberg and N. Mladenović. Degeneracy in the multi-source Weber problem. *Mathematical Programming*, 85:13–22, 1999.

[24] J. Brimberg, P. Hansen, and N. Mladenović. Attraction probabilities in variable neighborhood search. *4OR*, 8:181–194, 2010.

[25] P. Brucker, E. Burke, and S. Groenemeyer. A mixed integer programming model for the cyclic job-shop problem with transportation. *Discrete Applied Mathematics*, 160(13):1924–1935, 2012.

[26] S. Burer and A. N. Letchford. Non-convex mixed-integer nonlinear programming: a survey. *Surveys in Operations Research and Management Science*, 17(2):97–106, 2012.

[27] R. M. Burns and C. A. Gibson. Optimization of priority list for a unit commitment problem. *IEEE PES summer meeting*, 75:453–461, 1975.

[28] A. V. Cabot and A. P. Hurter Jr. An approach to zero-one integer programming. *Operations Research*, 16(6):1206–1211, 1968.

[29] R. W. Calvo. A new heuristic for the traveling salesman problem with time windows. *Transportation Science*, 34 (1):113–124, 2000.

[30] J. F. Campbell. Location and allocation for distribution systems with transshipments and transportation economies of scale. *Annals of Operations Research*, 40:77–99, 1992.

[31] J. F. Campbell. Integer programming formulations of discrete hub location problems. *European Journal of Operational Research*, 72: 387–405, 1994.

[32] J. F. Campbell. Hub location and the p-hub median problem. *Operations Research*, 44:923–935, 1996.

[33] W. B. Carlton and J. W. Barnes. Solving the travelling salesman problem with time windows using tabu search. *IEEE Transactions*, 28: 617–629, 1996.

[34] E. Carrizosa, N. Mladenović, and R. Todosijević. Sum-of-squares clustering on networks. *Yugoslav Journal of Operations research*, 21:157–161, 2011.

[35] E. Carrizosa, A. Al-Guwaizani, P. Hansen, and N. Mladenović. Degeneracy of harmonic means clustering. Working paper, 2013.

[36] E. Carrizosa, N. Mladenović, and R. Todosijević. Variable neighborhood search for minimum sum-of-squares clustering on networks. *European Journal of Operational Research*, 230(2):356–363, 2013.

[37] K. Chandram, N. Subrahmanyam, and M. Sydulu. Unit Commitment by improved pre-prepared power demand table and Muller method. *International Journal of Electrical Power & Energy Systems*, 33:106–114, 2011.

[38] C.-P. Cheng, C.-W. Liu, and C.-C. Liu. Unit commitment by Lagrangian relaxation and genetic algorithms. *IEEE Transactions on Power Systems*, 15(2):707–714, 2000.

[39] M. Chouman and T. Crainic. A MIP-tabu search hybrid framework for multicommodity capacitated fixed-charge network design. *Tecnical report CIRRELT–2010–31*, 2010.

[40] M. Chouman, T. Crainic, and B. Gendron. Commodity representations and cutset-based inequalities for multicommodity capacitated fixed-charge network design. *Tecnical report CIRRELT–2011-56*, 2011.

[41] C. Chung, H. Yu, and K. P. Wong. An advanced quantum-inspired evolutionary algorithm for unit commitment. *IEEE Trans Power Systems*, 26:847–854, 2011.

[42] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

[43] G. Collet, R. Andonov, N. Yanev, and J. Gibrat. Local protein threading by Mixed Integer Programming. *Discrete Applied Mathematics*, 159 (16):1707–1716, 2011.

[44] A. M. Costa, J.-F. Cordeau, and B. Gendron. Benders, metric and cutset inequalities for multicommodity capacitated network design. *Computational Optimization and Applications*, 42(3):371–392, 2009.

[45] A. M. Costa, J.-F. Cordeau, B. Gendron, and G. Laporte. Accelerating benders decomposition with heuristic master problem solutions. *Pesquisa Operacional*, 32(1):03–20, 2012.

[46] T. G. Crainic and M. Gendreau. Cooperative parallel tabu search for capacitated network design. *Journal of Heuristics*, 8(6):601–627, 2002.

[47] T. G. Crainic and M. Gendreau. *A scatter search heuristic for the fixed-charge capacitated network design problem.* Springer, 2007.

[48] T. G. Crainic, A. Frangioni, and B. Gendron. Multicommodity capacitated network design. In P. Soriano and B. Sanso, editors, *Telecommunications network planning*, pages 1–19. Kluwer Academics Publisher, Dordrecht, 1999.

[49] T. G. Crainic, M. Gendreau, and J. M. Farvolden. A simplex-based tabu search method for capacitated network design. *INFORMS Journal on Computing*, 12(3):223–236, 2000.

[50] T. G. Crainic, A. Frangioni, and B. Gendron. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics*, 112(1):73–99, 2001.

[51] T. G. Crainic, B. Gendron, and G. Hernu. A slope scaling/Lagrangean perturbation heuristic with long-term memory for multicommodity capacitated fixed-charge network design. *Journal of Heuristics*, 10(5): 525–545, 2004.

[52] T. G. Crainic, Y. Li, and M. Toulouse. A first multilevel cooperative algorithm for capacitated multicommodity network design. *Computers & Operations Research*, 33(9):2602–2622, 2006.

[53] I. Crévits, S. Elaoud, S. Hanafi, and C. Wilbaut. Diversification Generators for 0-1 Partial Solutions in Scatter Search. In M. Caserta and S. Voss, editors, *Metaheuristics (Operations Research / Computer Science Interfaces Series)*, pages 229–248. Springer, Berlin, 2011.

[54] R. Da Silva and S. Urrutia. A General VNS heuristic for the traveling salesman problem with time windows. *Discrete Optimization*, 7:203–211, 2010.

[55] I. G. S. Damousis, A. Bakirtziz, and P. Dokopoulos. A Solution to the Unit Commitment problem using integer coded genetic algorithm. *IEEE Trans Power Systems*, 19:1165–1172, 2004.

[56] E. Danna, E. Rothberg, L. Pape, and C. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102(1):71–90, 2005.

[57] G. Dantzig. *Programming in a linear structure*. Comptroller, USAF Washington, DC, 1948.

[58] G. Dantzig. *Linear Proxramming and Extensions*. Princeton University Press, Princeton, 1963.

[59] L. Di Gaspero and A. Schaerf. Neighborhood portfolio approach for local search applied to timetabling problems. *Journal of Mathematical Modelling and Algorithms*, 5(1):65–89, 2006.

[60] V. Dieu and W. Ongsakul. Enhanced augmented Lagrangian hopfield network for unit commitment. *IEEE Proc Gener Transm Distrib, 153*, pages 624–632, 2006.

[61] V. Dieu and W. Ongsakul. *Enhanced merit order and augmented Lagrangian hopfield network for ramp rate constrained unit commitment.* In Proc. of IEEE power system society meeting, Canada, 2006.

[62] V. Dieu and W. Ongsakul. Augmented Lagrange hopfield network based Lagrangian relaxation for unit commitment. *International Journal of Electrical Power & Energy Systems*, 33:522–530, 2011.

[63] J. D. Dillon, M. P. Walsh, and M. J. O. Malley. Initialization of the augmented Hopfield network for improved generator scheduling. *IEE Proc Gener Transm Distrib*, 149:593–599, 2002.

[64] J. Ebrahimi, S. Hosseinian, and G. Gharehpetian. Unit commitment problem solution using shuffled frog leaping algorithm. *IEEE Trans Power Systems*, 26:573–581, 2011.

[65] J. Eckstein and M. Nediak. Pivot, Cut and Dive: a heuristic for 0-1 mixed integer programming. *Journal of heuristics*, 13:471–503, 2007.

[66] M. M. El-Saadawi, M. A. Tantawi, and E. Tawfik. A fuzzy optimization-based approach to large scale thermal unit commitment. *Electric Power Systems Research*, 72:245–52, 2004.

[67] G. Erdogan, J. F. Cordeau, and G. Laporte. The Attractive Traveling Salesman Problem. *European Journal of Operational Research*, 203: 59–69, 2010.

[68] A. Ernst and M. Krishnamoorthy. Efficient algorithms for the uncapacitated single allocation p-hub median problem. *Location Science*, 4(3): 139–154, 1996.

[69] A. Ernst and M. Krishnamoorthy. An exact solution approach based on shortest paths for p-hub median problems. *INFORMS Journal on Computing*, 10:149–162, 1998.

[70] A. Ernst and M. Krishnamoorthy. Exact and heuristic algorithms for the uncapacitated multiple allocation p-hub median problems. *European Journal of Operational Research*, 4:100–112, 1998.

[71] R. Z. Farahani, M. Hekmatfar, A. B. Arabani, and E. Nikbakhsh. Hub location problems: A review of models, classification, solution techniques, and applications. *Computers & Industrial Engineering*, 64 (4):1096–1109, 2013.

[72] T. A. Feo and M. G. Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6:109–133, 1995.

[73] V. Filipović. An electromagnetism metaheuristic for the uncapacitated multiple allocation hub location problem. *Serdica Journal of Computing*, 5(3):261–272, 2011.

[74] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003.

[75] M. Fischetti and A. Lodi. Repairing mip infeasibility through local branching. *Computers and Operations Reaserch*, 35:1436–1445, 2008.

[76] M. Fischetti and M. Monaci. Proximity search for 0-1 mixed-integer convex programming. *Journal of Heuristics*, 20(6):709–731, 2014.

[77] M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.

[78] A. Frangioni and E. Gorgone. Generalized Bundle Methods for Sum-Functions with"Easy"Components: Applications to Multicommodity Network Design. *working paper*, 2013.

[79] A. Fréville and S. Hanafi. The Multidimensional 0–1 Knapsack Problem – Bounds and Computational Aspects. *Annals OR*, 139:195–227, 2005.

[80] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman San Francisco, 1979.

[81] M. Gendreau, A. Hertz, and L. G. New insertion and postoptimization procedures for the travelling salesman problem. *Operations Research*, 40:1086–1194, 1992.

[82] M. Gendreau, A. Hertz, G. Laporte, and M. Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operation Research*, 46 (3):330–335, 1998.

[83] B. Gendron and T. G. Crainic. Relaxations for multicommodity capacitated network design problems. Technical report, Univ., CRT, 1994.

[84] B. Gendron and M. Larose. Branch-and-price-and-cut for large-scale multicommodity capacitated fixed-charge network design. *EURO Journal on Computational Optimization*, 2(1-2):55–75, 2014.

[85] I. Ghamlouche, T. G. Crainic, and M. Gendreau. Cycle-based neighbourhoods for fixed-charge capacitated multicommodity network design. *Operations research*, 51(4):655–667, 2003.

[86] I. Ghamlouche, T. G. Crainic, and M. Gendreau. Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. *Annals of Operations research*, 131(1-4):109–133, 2004.

[87] F. Glover. A Multiphase Dual Algorithm for the Zero-One Integer Programming Problem. *Operations Research*, 13:879–919, 1965.

[88] F. Glover. A Note on Linear Programming and Integer Infeasibility. *Operations Research*, 16:1212–1216, 1968.

[89] F. Glover. Cut Search Methods in Integer Programming. *Mathematical Programming*, 3:86–100, 1972.

[90] F. Glover. Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences*, 8:156–166, 1977.

[91] F. Glover. Parametric branch and bound. *OMEGA, The International Journal of Management Science*, 6:1–9, 1978.

[92] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

[93] F. Glover. Scatter search and star-paths: beyond the genetic metaphor. *OR Spektrum*, 17:125–137, 1995.

[94] F. Glover. Tabu thresholding: Improved search by nonmonotonic trajectories. *ORSA Journal on Computing*, 7:426–442, 1995.

[95] F. Glover. A Template for Scatter Search and Path Relinking. pages 1–51. Artificial Evolution, Lecture Notes in Computer Science, Springer-Verlag, New York, 1998.

[96] F. Glover. Scatter Search and Path Relinking. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimisation*. McGraw-Hill, Boston, 1999.

[97] F. Glover. Parametric tabu-search for mixed integer programs. *Computers & Operations Research*, 33:2449–2494, 2006.

[98] F. Glover and S. Hanafi. Tabu search and finite convergence. *Discrete Applied Mathematics*, 119:3–36, 2002.

[99] F. Glover and S. Hanafi. Metaheuristic Search with Inequalities and Target Objectives for Mixed Binary Optimization-Part I: exploiting Proximity. *International Journal of Applied Metaheuristic Computing (IJAMC)*, 1:1–15, 2010.

[100] F. Glover and S. Hanafi. Metaheuristic Search with Inequalities and Target Objectives for Mixed Binary Optimization-Part II: Exploiting Reaction and Resistance. *International Journal of Applied Metaheuristic Computing (IJAMC)*, 1(2):1–17, 2010.

[101] F. Glover and M. Laguna. General Purpose Heuristics for Integer Programming-Part II. *Journal of Heuristics*, 3:161–179, 1997.

[102] F. Glover and M. Laguna. Fundamentals of Scatter Search and Path Relinking. *Control and Cybernetics*, 29:653–684, 2000.

[103] F. Glover, C. McMillan, and R. Glover. A heuristic programming approach to the employee scheduling problem and some thoughts on managerial robots. *Journal of Operations Management*, 4(2):113–128, 1984.

[104] F. Glover, E. Taillard, and D. de Werra. A user's guide to tabu search. *Annals of Operations Research*, 41:3–28, 1993.

[105] F. Glover, M. Laguna, and R. Marti. Scatter Search. In A. Ghosh and S. Tsutsui, editors, *Theory and Applications of Evolutionary Computation: Recent Trends*. Springer-Verlag, New York, 2000.

[106] F. Glover, A. Løkketangen, and D. L. Woodruff. Scatter search to generate diverse MIP solutions. In *Computing Tools for Modeling, Optimization and Simulation*, pages 299–317. Springer, 2000.

[107] A. Grigoriev, J. Van De Klundert, and F. Spieksma. Modeling and solving the periodic maintenance problem. *European journal of operational research*, 172(3):783–797, 2006.

[108] S. L. Hakimi. Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations Research*, 13:462–475, 1965.

[109] S. Hanafi. On the Convergence of Tabu Search. *Journal of Heuristics*, 7:47–58, 2001.

[110] S. Hanafi and A. Fréville. An Efficient Tabu Search Approach for the 0-1 Multidimensional Knapsack Problem. *European Journal of Operational Research*, 106:659–675, 1998.

[111] S. Hanafi and C. Wilbaut. Scatter search for the 0-1 multidimensional knapsack problem. *Journal of Mathematical Modelling and Algorithms*, 7:143–159, 2008.

[112] S. Hanafi and C. Wilbaut. Improved convergent heuristics for the 0-1 multidimensional knapsack problem. *Annals OR*, 183:125–142, 2011.

[113] S. Hanafi, J. Lazić, and N. Mladenović. Variable Neighbourhood Pump Heuristic for 0–1 Mixed Integer Programming Feasibility. *Electronic Notes in Discrete Mathematics*, 36:759–766, 2010.

[114] S. Hanafi, J. Lazić, N. Mladenović, C. Wilbaut, and I. Crévits. Hybrid Variable Neighbourhood Decomposition Search for 0–1 Mixed Integer Programming Problem. *Electronic Notes in Discrete Mathematics*, 36: 883–890, 2010.

[115] P. Hansen. The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming. *Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy*, 1986.

[116] P. Hansen and B. Jaumard. Cluster Analysis and Mathematical Programming. *Mathematical Programming*, 79:191–215, 1997.

[117] P. Hansen and N. Mladenović. Simultaneous static unit commitment and economic dispatch by dynamic programming. *Cahiers du GERAD, G-96-26*, 1996.

[118] P. Hansen and N. Mladenović. Variable neighborhood search for the p-median. *Location Science*, 5:207–226, 1997.

[119] P. Hansen and N. Mladenović. J-Means: A new local search heuristic for minimum sum-of-squares clustering. *Pattern Recognition*, 34:405–413, 2001.

[120] P. Hansen and N. Mladenović. A separable approximation dynamic programming algorithm for economic dispatch with transmission losses. *Yugoslav Journal of Operations Research,*, 12:157–166, 2002.

[121] P. Hansen and N. Mladenović. First vs. best improvement: An empirical study. *Discrete Applied Mathematics*, 154(5):802–817, 2006.

[122] P. Hansen, N. Mladenović, and D. Perez-Britos. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4):335–350, 2001.

[123] P. Hansen, E. Ngai, B. Cheung, and N. Mladenović. Analysis of Global k-Means, an Incremental Heuristic for Minimum Sum-of-Squares Clustering. *Journal of Classification*, 22:287–310, 2005.

[124] P. Hansen, N. Mladenović, and D. Urosević. Variable neighborhood search and local branching. *Computers & Operations Research*, 33(10): 3034–3045, 2006.

[125] P. Hansen, J. Brimberg, D. Uroševic, and N. Mladenovic. Primal-dual variable neighborhood search for the simple plant-location problem. *INFORMS Journal on Computing*, 19(4):552–564, 2007.

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

[127] M. Hewitt, G. L. Nemhauser, and M. W. Savelsbergh. Combining exact and heuristic approaches for the capacitated fixed-charge network flow problem. *INFORMS Journal on Computing*, 22(2):314–325, 2010.

[128] M. Hewitt, G. Nemhauser, and M. W. Savelsbergh. Branch-and-price guided search for integer programs with an application to the multi-commodity fixed-charge network flow problem. *INFORMS Journal on Computing*, 25(2):302–316, 2013.

[129] M. J. Hodgson, G. Laporte, and F. Semet. A covering tour model for planning mobile health care facilities in Suhum District, Ghana. *Journal of Regional Science*, 38:621–638, 1998.

[130] K. Holmberg and D. Yuan. A Lagrangian heuristic based branch-and-bound approach for the capacitated network design problem. *Operations Research*, 48(3):461–481, 2000.

[131] J. N. Hooker, R. Garfinkel, and C. Chen. Finite dominating sets for network location problems. *Operations Research*, 39:100–118, 2001.

[132] K. Y. Huang, H. T. Yang, and C. L. Huang. A new thermal unit commitment approach using constraint logic programming. *IEEE Trans Power Systems*, 13:936–945, 1998.

[133] A. Ilić, D. Urošević, J. Brimberg, and N. Mladenović. A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. *European Journal of Operational Research*, 206 (2):289–300, 2010.

[134] A. Imai, E. Nishimura, and J. Current. A Lagrangian relaxation-based heuristic for the vehicle routing with full container load. *European journal of operational research*, 176(1):87–105, 2007.

[135] R. A. Jabr. Rank-constrained semidefinite program for unit commitment. *International Journal of Electrical Power & Energy Systems*, 47: 13–20, 2013.

[136] S. D. Jena, J.-F. Cordeau, and B. Gendron. Lagrangian Heuristics for Large-Scale Dynamic Facility Location with Generalized Modular Capacities. *Tecnical report CIRRELT—2014–21*, 2014.

[137] Y. W. Jeong, J. B. Park, S. H. Jang, and K. Lee. A new quantum inspired binary pso: Application to unit commitment problems for power systems. *IEEE Trans Power Systems*, 25:1486–1495, 2010.

[138] D. S. Johnson and L. McGeoch. The traveling salesman problem: a case study in local optimization. In E. Aarts and J. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley and Sons, New York, 1997.

[139] K. A. Juste, H. Kita, E. Tanaka, and J. Hasegawa. An evolutionary programming solution to the unit commitment problem. *IEEE Trans Power Systems*, 14:1452–1459, 1999.

[140] N. Katayama and S. Yurimoto. Combining Capacity Scaling and Local Branch Approaches for the Logistics Network Design Problem. *21st International Conference on Production Research*, 2013.

[141] N. Katayama, M. Chen, and M. Kubo. A capacity scaling heuristic for the multicommodity capacitated network design problem. *Journal of computational and applied mathematics*, 232(1):90–101, 2009.

[142] A. Kazarlis, A. G. Bakirtzis, and V. Petridis. A genetic algorithm solution to the unit commitment problem. *IEEE Trans Power Systems*, 11:83–92, 1996.

[143] M. Khemakhem, B. Haddar, K. Chebil, and S. Hanafi. A Filter-and-Fan Metaheuristic for the 0-1 Multidimensional Knapsack Problem. *Int. J. of Applied Metaheuristic Computing*, 3:43–63, 2012.

[144] G. Kliewer and L. Timajev. Relax-and-cut for capacitated network design. In *Algorithms–ESA 2005*, pages 47–58. Springer, 2005.

[145] J. Kratica. An electromagnetism-like metaheuristic for the uncapacitated multiple allocation p-hub median problem. *Computers & Industrial Engineering*, 66(4):1015–1024, 2013.

[146] J. Kratica, Z. Stanimirović, D. Tosić, and V. Filipović. Two genetic algorithms for solving the uncapacitated single allocation p-hub median problem. *European Journal of Operational Research*, 182(1):15–28, 2007.

[147] M. Labbé, D. Peeters, and J. F. Thisse. Location on networks. In M. O. Ball, M. C. L. Magnanti, T. L., and G. L. Nemhauser, editors, *Handbooks in OR & MS*, pages 551–624. Elsevier, Amsterdam, 1995.

[148] M. Laguna and R. Marti. *Scatter Search–Methodology and Implementations in C.* Kluwer Academic Publishers, Norwell, MA, 2003.

[149] M. Laguna and R. Martí. Experimental testing of advanced scatter search designs for global optimization of multimodal functions. *Journal of Global Optimization*, 33(2):235–255, 2005.

[150] T. Lau, C. Chung, K. Wong, T. Chung, and S. Ho. Quantum-inspired evolutionary algorithm approach for unit commitment. *IEEE Trans Power Systems*, 24:1503–1512, 2009.

[151] J. Lazić, S. Hanafi, N. Mladenović, and D. Urosević. Variable neighbourhood decomposition search for 0–1 mixed integer programs. *Computers & Operations Research*, 37:1055–1067, 2010.

[152] J. Lazić, R. Todosijević, S. Hanafi, and N. Mladenović. Variable and single neighbourhood diving for MIP feasibility. *Yugoslav Journal of Operations Research*, 2014. doi: 10.2298/YJOR140417027L.

[153] J. Lee and S. Leyffer. *Mixed integer nonlinear programming*, volume 154. Springer Science & Business Media, 2011.

[154] A. Lokketangen and F. Glover. Solving zero-one mixed integer programming problems using tabu search. *European Journal of Operational Research*, 106:624–658, 1998.

[155] M. Lopez-Ibanez and C. Blum. Beam-ACO for the travelling salesman problem with time windows. *Computers & operations research*, 37(9): 1570–1583, 2010.

[156] R. F. Love, J. G. Morris, and G. O. Wesolowski. *Facilities location: models and methods*. Elsevier Science Publishing Co., New York, 1988.

[157] P. G. Lowery. Generating unit commitment by dynamic programming. *IEEE Trans Power Systems*, 85:422–426, 1966.

[158] Z. Lü, J.-K. Hao, and F. Glover. Neighborhood analysis: a case study on curriculum-based course timetabling. *Journal of Heuristics*, 17(2): 97–118, 2011.

[159] T. L. Magnanti and R. T. Wong. Network design and transportation planning: Models and algorithms. *Transportation science*, 18(1):1–55, 1984.

[160] V. Maniezzo, T. Stützle, and S. Voß. Matheuristics, Annals of information systems 10, 2010.

[161] R. Marti, F. Glover, and M. Laguna. Principles of scatter search. *European Journal of Operational Research*, 169:359–372, 2006.

[162] S. N. Mhanna and R. Jabr. Application of semidefinite programming relaxation and selective pruning to the unit commitment problem. *Electric Power Systems Research*, 90:85–92, 2012.

[163] M. Milanović. A new evolutionary based approach for solving the uncapacitated multiple allocation p-hub median problem. In G. X. et al., editor, *Soft Computing in Industrial Applications*, pages 81–88. Springer-Verlag, AISC75. Berlin, 2010.

[164] S. Mitrović-Minić and A. Punnen. Very large-scale variable neighborhood search for the multi-resource generalized assignment problem. *Discrete Optimization*, 6(4):370–377, 2009.

[165] S. Mitrović-Minić and A. Punnen. Variable Intensity Local Search. In V. Maniezzo, T. Stützle, and S. Voß, editors, *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*, pages 245–252. Springer, 2009.

[166] S. Mitrović-Minić and A. P. Punnen. Very large-scale variable neighborhood search for the generalized assignment problem. *Journal of Interdisciplinary Mathematics*, 11(5):653–670, 2008.

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

[168] N. Mladenović, F. Plastria, and D. Urošević. Reformulation descent applied to circle packing problems. *Computers & Operations Research*, 32(9):2419–2434, 2005.

[169] N. Mladenović, R. Todosijević, and D. Urošević. An efficient general variable neighborhood search for large travelling salesman problem with time windows. *Yugoslav Journal of Operations Research*, 23(1):19–31, 2013.

[170] N. Mladenović, R. Todosijević, and D. Urošević. Two level General variable neighborhood search for Attractive traveling salesman problem. *Computers and Operations Research*, 52:341–348, 2014.

[171] M. Moghimi Hadji and B. Vahidi. A solution to the unit commitment problem using imperialistic competition algorithm. *IEEE Trans Power Systems*, 27:117–124, 2012.

[172] L. Mönch, J. W. Fowler, S. Dauzère-Pérès, S. J. Mason, and O. Rose. A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. *Journal of Scheduling*, 14 (6):583–599, 2011.

[173] J. W. Ohlmann and B. W. Thomas. A compressed-annealing heuristic for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 19:80–90, 2007.

[174] M. E. O'Kelly. A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research*, 32(3): 393–404, 1987.

[175] W. Ongsakul and N. Petcharaks. Unit commitment by enhanced adaptive Lagrangian relaxation. *IEEE Trans Power Systems*, 19:620–628, 2004.

[176] J. R. Oppong and M. Hodgson. Spatial accessibility to health care facilities in Suhum District, Ghana. *The Professional Geographer*, 46: 199–209, 1994.

[177] C. K. Pang, G. B. Sheble, and F. Albu. Evaluation of dynamic programming based methods and multiple area representation for thermal unit commitment. *IEEE Trans Power Appl Syst*, 100:1212–1218, 1981.

[178] D. Paraskevopoulos, T. Bektas, T. Crainig, and C. Potts. A Cycle Based Evolutionary Algorithm for the Fixed Charge Capacitated Multi

Commodity Network Design Problem. *Tecnical report CIRRELT–2013–08*, 2013.

[179] D. Paraskevopoulos, T. Bektas, T. Crainig, and C. Potts. A Cycle Based Evolutionary Algorithm for the Fixed Charge Capacitated Multi Commodity Network Design Problem. *Tecnical report CIRRELT–2014–36*, 2014.

[180] J. Peiró, A. Corberan, and R. Marti. GRASP for the uncapacitated r-allocation p-hub median problem. *Computers & Operations Research*, 43:50–60, 2014.

[181] D. Pisinger. An expanding-core algorithm for the exact 0-1 knapsack problem. *European Journal of Operational Research*, 87(1):175–187, 1995.

[182] J. Puchinger and G. Raidl. Bringing order into the neighborhoods: Relaxation guided variable neighborhood search. *Journal of Heuristics*, 14(5):457–472, 2008. ISSN 1381-1231.

[183] J. Puchinger, G. Raidl, and U. Pferschy. The core concept for the multidimensional knapsack problem. *Lecture Notes in Computer Science*, 3906:195–208, 2006.

[184] R. Quan, J. Jian, and Y. Mu. Tighter relaxation method for unit commitment based on second-order cone programming and valid inequalities. *International Journal of Electrical Power & Energy Systems*, 55:82–90, 2014.

[185] C. C. A. Rajan and M. R. Mohan. An evolutionary programming based Tabu search method for solving the unit commitment problem. *IEEE Trans Power Syst*, 19:577–585, 2004.

[186] J. G. Rakke, M. Christiansen, K. Fagerholt, and G. Laporte. The Travelling Salesman Problem with Draft Limits. *Computers & Operations Research*, 39:2162–2167, 2012.

[187] C. Rego and F. Glover. Ejection chain and filter-and-fan methods in combinatorial optimization. *Annals of Operations Research*, 175(1): 77–105, 2010.

[188] C. Rego, D. Gamboa, F. Glover, and C. Osterman. Traveling salesman problem heuristics: leading methods, implementations and latest advances. *European Journal of Operational Research*, 211(3):427–441, 2011.

[189] G. Reinelt. TSPLIB - A traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991.

[190] M. G. C. Resende, C. C. Ribeiro, F. Glover, and R. Marti. Scatter Search and Path-Relinking: Fundamentals, Advances and Applications. *Handbook of Metaheuristics, International Series in Operations Research & Management Science*, 146:87–107, 2010.

[191] I. Rodríguez-Martín and J. J. Salazar-González. A local branching heuristic for the capacitated fixed-charge network design problem. *Computers & Operations Research*, 37(3):575–581, 2010.

[192] K. Roy. Solution of unit commitment problem using gravitational search algorithm. *International Journal of Electrical Power & Energy Systems*, 53:85–94, 2013.

[193] A. Y. Saber and A. M. Alshareef. Scalable unit commitment by memory-bounded ant colony optimization with A* local search. *International Journal of Electrical Power & Energy Systems*, 30(6):403–414, 2008.

[194] A. Y. Saber, T. Senjyu, T. Miyagi, N. Urasaki, and T. Funabashi. Fuzzy unit commitment scheduling using absolutely stochastic simulated annealing. *IEEE Trans Power Syst*, 21:955–964, 2006.

[195] A. Y. Saber, T. Senjyu, T. Miyagi, N. Urasaki, and T. Funabashi. Unit commitment by heuristics and absolutely stochastic simulated annealing. *IET Gener Transm Distrib*, 1:234–243, 2007.

[196] A. Y. Saber, T. Senjyu, A. Yona, and T. Funabashi. Unit commitment computation by fuzzy adaptive particle swarm optimisation. *IET Gener Transm Distrib*, 1:456–465, 2007.

[197] A. Y. Saber, T. Senjyu, A. Yona, N. Urasaki, and T. Funabashi. Fuzzy unit commitment solution – a novel twofold simulated annealing approach. *Electr Power Syst Res*, 77:1699–1712, 2007.

[198] L. H. Sacchi and V. A. Armentano. A computational study of parametric tabu search for 0–1 mixed integer programs. *Computers & Operations Research*, 38(2):464–473, 2011.

[199] R. M. Saltzman and F. S. Hillier. A heuristic ceiling point algorithm for general integer linear programming. *Management Science*, 38(2): 263–283, 1992.

[200] H. Sasaki, M. Watanabe, and R. Yokoyama. A solution method of unit commitment by artificial neural networks. *IEEE Trans Power Systems*, 7:974–981, 1992.

[201] M. Sellmann and G. Kliewe. Lagrangian cardinality cuts and variable fixing for capacitated network design. In *Algorithms–ESA 2002*, pages 845–858. Springer, 2002.

[202] T. Senjyu, H. Yamashiro, K. Shimabukuro, K. Uezato, and T. Funabashi. A unit commitment problem by using genetic algorithm based on characteristic classification. *IEEE/Power Eng Soc Winter Meet*, 1: 58–63, 2002.

[203] I. J. Silva, S. J. Carneiro, E. J. De Oliveira, J. L. R. Pereira, P. A. N. Garcia, and A. Marcato. A Lagrangian multiplier based sensitive index to determine the unit commitment of thermal units. *International Journal of Electrical Power & Energy Systems*, 30:504–510, 2008.

[204] A. Soyster, B. Lev, and W. Slivka. Zero-one programming with many variables and few constraints. *European Journal of Operational Research*, 2(3):195–201, 1978.

[205] D. Srinivasan and J. Chazelas. A priority list based evolutionary algorithm to solve large scale unit commitment problem. *In: International conference on power system technology Powercon 2004, Singapore*, pages 21–24, 2004.

[206] L. Sun, Y. Zhang, and C. Jiang. A matrix real-coded genetic algorithm to the unit commitment problem. *Electric Power Systems Research*, 76:716–728, 2006.

[207] K. S. Swarup and S. Yamashiro. Unit commitment solution methodology using genetic algorithm. *IEEE Trans Power Syst*, 17:87–91, 2002.

[208] R. Todosijević, M. Mladenović, S. Hanafi, and I. Crévits. VNS based heuristic for solving the Unit Commitment problem. *Electronic Notes in Discrete Mathematics*, 39:153–160, 2012.

[209] R. Todosijević, R. Benmanosur, S. Hanafi, N. Mladenović, and A. Artiba. Nested general variable neighborhood search for the periodic maintenance problem. *European Journal of Operational Research, submitted*, 2014.

[210] R. Todosijević, A. Mjirda, M. Mladenović, S. Hanafi, and B. Gendron. A general variable neighborhood search variants for the travelling salesman problem with draft limits. *Optimization Letters*, 2014. doi: 10.1007/s11590-014-0788-9.

[211] R. Todosijević, M. Mladenović, S. Hanafi, N. Mladenović, and I. Crévts. Adaptive general variable neighborhood search heuristics for solving unit commitment problem. *Interanational Journal of Electrical Power and Energy Systems, submitted*, 2014.

[212] R. Todosijević, S. Hanafi, D. Urošević, B. Jarboui, and B. Gendron. Variable Neighborhood Search based heuristic for the Swap-Body Vehicle Routing Problem. *submitted*, 2015.

[213] R. Todosijević, D. Urošević, N. Mladenović, and S. Hanafi. A general variable neighborhood search for solving the uncapacitated r-allocation p-hub median problem. *Optimization Letters*, 2015. doi: 10.1007/s11590-015-0867-6.

[214] F. M. B. Toledo and V. A. Armentano. A Lagrangian-based heuristic for the capacitated lot-sizing problem in parallel machines. *European Journal of Operational Research*, 175(2):1070–1083, 2006.

[215] H. Tuy. Concave Programming Under Linear Constraints. *Soviet Mathematics*, pages 1437–1440, 1964.

[216] M. Vasquez and J. K. Hao. A Hybrid Approach for the 0–1 Multidimensional Knapsack problem. *RAIRO - Operations Research*, 35:415–438, 2001.

[217] M. Vasquez and Y. Vimont. Improved results on the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, 165:70–81, 2005.

[218] VeRoLog. Swap-Body Vehicle Routing Problem, 2014. http://verolog.deis.unibo.it/news-events/general-news/verolog-solver-challenge-2014.

[219] A. Viana and J. P. Pedroso. A new MILP-based approach for Unit Commitment in power production planning. *International Journal of Electrical Power & Energy Systems*, 44:997–1005, 2013.

[220] T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins. Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, 231(1):1–21, 2013.

[221] C. Wilbaut and S. Hanafi. New convergent heuristics for 0–1 mixed integer programming. *European Journal of Operational Research*, 195: 62–74, 2009.

[222] C. Wilbaut, S. Hanafi, and S. Salhi. A survey of effective heuristics and their application to a variety of knapsack problems. *IMA Journal of Management Mathematics*, 19:227–244, 2008.

[223] C. Wilbaut, S. Salhi, and S. Hanafi. An iterative variable-based fixation heuristic for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 199(2):339–348, 2009.

[224] L. Wolsey and G. Nemhauser. Integer and Combinatorial Optimization, 1999.

[225] A. J. Wood and B. F. Wollenberg. *Power generation, operation and control, 2nd revised edition.* Wiley, New York, 1996.

[226] M. Yaghini, M. Rahbar, and M. Karimi. A hybrid simulated annealing and column generation approach for capacitated multicommodity network design. *Journal of the Operational Research Society*, 64(7): 1010–1020, 2012.

[227] H. Yaman. Allocation strategies in hub networks. *European Journal of Operational Research*, 211(3):442–451, 2011.

[228] R. D. Young. Hypercylindrically deduced cuts in zero-one integer programs. *Operations Research*, 19(6):1393–1405, 1971.

[229] B. Zhao, C. X. Guo, B. R. Bai, and Y. J. Cao. An improved particle swarm optimization algorithm for unit commitment. *International Journal of Electrical Power & Energy Systems*, 44:432–512, 2006.