J Heuristics (2017) 23:165–206
DOI 10.1007/s10732-017-9336-y

Mathematical programming based heuristics for the 0–1 MIP: a survey

Saïd Hanafi[1] · **Raca Todosijević**[1]

Received: 10 January 2016 / Revised: 26 April 2017 / Accepted: 4 May 2017 / Published online: 19 May 2017

Abstract The 0–1 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. This paper provides a survey of heuristics based on mathematical programming for solving 0–1 mixed integer programs (MIP). More precisely, we focus on the stand-alone heuristics for 0–1 MIP as well as those heuristics that use linear programming techniques or solve a series of linear programming models or reduced problems, deduced from the initial one, in order to produce a high quality solution of a considered problem. Our emphasis will be on how mathematical programming techniques can be used for approximate problem solving, rather than on comparing performances of heuristics.

Keywords Mathematical programming · Heuristics · Metaheuristics · 0–1 Mixed integer program

1 Introduction

The 0–1 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:

✉ Saïd Hanafi
said.hanafi@univ-valenciennes.fr

Raca Todosijević
racatodosijevic@gmail.com

[1] LAMIH UMR CNRS 8201 - Université de Valenciennes, 59313 Valenciennes Cedex 9, France

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

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, and otherwise may be infinite). 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., $X = \{x \in \mathbb{R}^n : Ax \leq b;\ 0 \leq x_j \leq U_j,\ \ j \in N;\ \ x_j \in \{0, 1\},\ \ j \in \mathcal{I}\}$. 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 well-designed and carefully conceived heuristics is not only their ability to produce high-quality solutions in a 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 as well as to accelerate the exploration of the tree. Consequently, this survey focuses on general heuristics based on mathematical programming techniques for 0–1 MIP problems, as opposed to those that exploit problem structure (e.g., Lagrangian based heuristics (as in Beasley 1993; Toledo and Armentano 2006; Holmberg and Yuan 2000; Jena et al. 2014; Imai et al. 2007), or heuristics proposed for various special combinatorial structures, like scheduling and location problems, the traveling salesman problem etc. (see e.g., Rego et al. 2011; Farahani et al. 2013; Vidal et al. 2013; Mönch et al. 2011; Ball 2011). 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 Bonami et al. 2012; Burer and Letchford 2012; Lee and Leyffer 2011; Berthold 2014; Fischetti and Lodi 2011 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 liner 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 (LP relaxation, MIP relaxation, simplex pivot, branch-and-bound, convexity cut, pseudo cut, decomposition, etc.) can be used for approximate problem solving, rather than on comparing the performance of heuristics. To the best of our knowledge there is not a recent survey on the heuristics for 0–1 MIP.

The rest of the paper is organized as follows. In Sect. 2, 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 3 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 4 is devoted to so-called pump heuristics which purpose is to create a first feasible solution of the considered MIP. Section 5 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. Section 7 provides a classification and summary of main components of MIP heuristics. Finally, Sect. 8 concludes the paper and indicates possible directions for future work.

Here we describe notation used throughout the paper. 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 $\alpha + 0.5$ (i.e., the greatest integer $\leq \alpha + 0.5$). Furthermore, 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, by defining each component as $near(x)_j = near(x_j) = \lfloor x_j + 0.5 \rfloor$. We first define a measure $u(\alpha)$ of integer infeasibility for a assigning a value α to a variable by the rule $u_r(\alpha) = |\alpha - near(\alpha)|^r$, where the exponent r is a 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 a function takes the value 0 if α is an integer, 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 $j \notin J$. The 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 $\delta(J, x, y) = \sum_{j \in J} | x_j - x'_j |$ (hence, $\delta(\mathcal{I}, x, x') = \delta(x, x')$). 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.

2 Pivoting heuristics

In this section we review heuristics that make use of pivot moves in order to solve a MIP problem at hand. By applying a pivot move such heuristics actually move from an extreme point to an adjacent one. Such heuristics are inspired by the observation 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. Namely, 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; Glover 1968).

The bounded simplex method proposed by Dantzig (1948, 1963) is an efficient method to solve the LP- relaxation of the MIP problem by systematically exploring extreme points of the 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

$$x^j = x^0 - \theta_j D_j \text{ for } j \in \overline{B} \tag{2}$$

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

$$D_{kj} = \begin{cases} ((A^B)^{-1}A)_{kj} & \text{if } k \in B \\ \xi & \text{if } k = j \\ 0 & \text{if } k \in \overline{B} - \{j\} \end{cases} \tag{3}$$

where A^B represents the matrix obtained from the 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. In what follows, the set of extreme points adjacent to an extreme point x^0 will be denoted by

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

The heuristics which will be reviewed in this section are: Pivot and Complement (Balas and Martin 1980), Enhanced Pivot and Complement (Aboudi et al. 1989), Pivot and Shift (Balas et al. 2004), Pivot and Tabu (Lokketangen and Glover 1998), and Pivot Cut and Dive (Eckstein and Nediak 2007) heuristics.

2.1 Pivot and complement heuristic

Balas and Martin (1980) proposed a *Pivot and Complement* (P&C) heuristic for a pure 0–1 MIP (i.e., $\mathcal{I} = N$). 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 a feasible 0–1 solution, in order to find a first MIP feasible solution, the 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. Let $\mathcal{S}$ be the set of slack variables, 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

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

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

```
   Function FS();
1  Solve LP relaxation to obtain an optimal LP basic solution $\overline{x}$;
2  if $\overline{x} \in \{0,1\}^n$ then
       $Stop = True$; // it is optimal;
   else
3      Set $x = \overline{x}$;$Stop = False$;
   end
4  while $Stop = False$ do
5      while $\mathcal{N}_1(x) \cup \mathcal{N}_2(x) \neq \emptyset$ and $x \notin \{0,1\}^n$ do
6          if $\mathcal{N}_1(x) \neq \emptyset$ then
               select $x' = argmax\{cx : x \in \mathcal{N}_1(x)\}$;
           else
7              if $\mathcal{N}_2(x) \neq \emptyset$ then
                   select $x' \in \mathcal{N}_2(x)$;
               end
           end
8          set $x = x'$;
       end
9      if $\{x, near(x), \lfloor x \rfloor\} \cap X \neq \emptyset$ then
           $Stop = True$; Break;
       end
10     select $x' = argmin\{\mu(y) : y \in \mathcal{N}_3(x)\}$; set $x = x'$;
11     while $\mathcal{N}_4^1(x) \cup \mathcal{N}_4^2(x) \neq \emptyset$ and $x$ infeasible do
12         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
13         set $x = x'$;
       end
14     if $x$ infeasible then
           $Stop = Fail$;
       else
15         if $\{x, near(x), \lfloor x \rfloor\} \cap X \neq \emptyset$ then
               $Stop = True$;
           end
       end
   end
```

If Pivot and Complement heuristic succeeds in finding a MIP feasible solution, an improvement phase is launched to possibly improve the obtained solution. The improvement phase Algorithm 2 used inside the Pivot and Complement heuristic is based on variable fixing and complementing. Note that the improvement phase may be seen as a variable neighborhood descent approach (Hansen et al. 2010). 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 optimal and thus the variable is fixed. The complementing step 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 of variables $J \subset \mathcal{I}$ is a candidate for complementing if

$$\sum_{j \in J}(1 - 2x_j)c_j \geq 1 \tag{4}$$

and

$$\sum_{j \in J}(1 - 2x_j)a_{ij} \leq b_i - \sum_{j \notin J} x_j a_{ij}, \forall i \in \{1, \ldots m\}. \tag{5}$$

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

$$\mathcal{N}'^{k}_{4}(x) = \{\overline{x(J)} : J \subset \mathcal{I}, |J| = k, A\overline{x(J)} \leq b \text{ and } J \text{ satisfy (4) and (5)}\}$$

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 $\mathcal{N}'^{1}_{4}(x) \neq \emptyset$ then
           select $x' = argmax\{cx : x \in \mathcal{N}'^{1}_{4}(x)\}$;
           set $x = x'$;
           continue;
       end
5      if $\mathcal{N}'^{2}_{4}(x) \neq \emptyset$ then
           select $x' \in \mathcal{N}'^{2}_{4}(x)$;
           set $x = x'$;
           continue;
       end
6      if $\mathcal{N}'^{3}_{4}(x) \neq \emptyset$ then
           select $x' \in \mathcal{N}'^{3}_{4}(x)$;
           set $x = x'$;
           continue;
       end
       $Stop = True$;
   end
```

Aboudi et al. (1989) proposed an *enhancement of the 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.

Balas et al. (2004) 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. The search phase attempts to construct a feasible solution by examining three neighborhood structures based on the pivot moves and a procedure inspired by local branching. If no replacement occurs as a result of searching one of the neighborhoods, the search for a feasible solution resumes by exploring a specially defined 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. Then the search is done by using a MIP solver in the neighborhood defined as follows:

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

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 improves 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:

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

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

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

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.

2.2 Pivot and tabu heuristic

Tabu search (TS) (Glover 1986) 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 1977a) . Aboudi and Jörnsten (1994) proposed a tabu search where the P&C heuristic is used as subroutine and the tabu conditions are implemented by serially adding constraints that eliminates the current local optimum. Løkketangen et al. (1994) embeded TS framework within the Pivot and Complement heuristic.

Since an optimal solution for the 0–1 MIP problem may be found at an extreme point of the LP feasible set, Lokketangen and Glover (1998) 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 the 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:

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

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. 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 problems, examination of entire neighborhood is not possible in a reasonable amount of time. Therefore, the strategies described in Glover et al. (1993) and in Glover (1995b) 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

$$\begin{aligned}\Delta v(h) &= cx^h - cx^0\\ \Delta u(h) &= u_r(x^0, \mathcal{I}) - u_r(x^h, \mathcal{I}).\end{aligned}$$

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.
The 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 when replacing the current solution by one neighboring. After that the best solution is determined using 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 (of 4 types previously mentioned) ratio of $\Delta v(h)$ and $\Delta u(h)$ is calculated. After that the best solution of each type is determined, and finally the best one among them 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 $Frequency(j)/Current\_Iteration$.

2.3 Pivot cut and dive heuristic

Eckstein and Nediak (2007) 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 a 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 the previously defined merit function the following statements holds:

Property 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 improve 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 the 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 have 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 the 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 by adding a 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 is executed: a recursive, depth-first diving operation that seeks to recover feasibility. If feasibility is successfully repaired the rounding procedure is repeated, while otherwise, the procedure fails (no feasible solution of the (sub)problem is found).

2.4 Summary

The Pivot and Complement heuristic is implemented as a general heuristic in the XMP/ZOOM software (Marsten 1987). The major drawback of both Pivot and Complement and Pivot and Shift heuristics is that they do not guarantee to produce a feasible solution. As shown in Balas et al. (2004), Pivot and Shift heuristic is able to enhance the performance of a MIP solver if the MIP solver is joined with Pivot and Shift heuristic. Namely, in Balas et al. (2004) Pivot and Shift were joined to XPRESS MIP solver, and in that way better results in shorter time were obtained comparing to the case when XPRESS MIP solver is used alone. Since above heuristics had not been tested on the same set of test instances, it is hard to say which one is the best (For example, tabu search algorithms are tested only on instances of multidimensional knapsack problem, while Pivot, Cut and Dive heuristic is tested only on 49 MIPLIB 3.0 instances without any comparison with existing methods). However, Lokketangen and Glover (1998) showed that the proposed tabu search performed better than Pivot and Complement, but the computational results are limited to relatively small multidimensional knapsack problems. In addition, it is evident that Enhanced Pivot and Complement heuristic outperforms the classical one as already pointed out. Among the considered heuristics, only Pivot and Shift heuristic may be used for solving general mixed integer programming problems.

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

```
Function Pivot&cut&dive($P$);
1  Solve the LP relaxation of $P$ to obtain an optimal LP basic solution $\overline{x}$;
2  $Stop = False$;
3  $Integer = False$;
4  Set $x = \overline{x}$;
5  while Stop=False and Integer = False do
6      if $x$ MIP feasible then
           $Integer = True$;
       end
7      if $\mathcal{N}''_1(x) \neq \emptyset$ then
8          select best $x' \in \mathcal{N}''_1(x)$ according to imposed rule;
9          set $x = x'$;
10         continue;
       end
11     if $\mathcal{N}''_2(x) \neq \emptyset$ then
12         select best $x' \in \mathcal{N}''_2(x)$ according to imposed rule;
13         set $x = x'$;
14         continue;
       end
15     if $\mathcal{N}''_3(x) \neq \emptyset$ then
16         select first $x' \in \mathcal{N}''_3(x)$ according to imposed rule;
17         set $x = x'$;
18         continue;
       end
19     $Stop = True$;
   end
20 if Integer=False then
21     $Q = (P|$ convexity cut);
22     if $Q$ is feasible then
23         Pivot&cut&dive($Q$);
       else
24         Try to repair feasibility of $Q$ ;
25         if Feasibility repaired then
26             Pivot&cut&dive($Q$);
           else
27             Report failure;
           end
       end
   end
```

3 Pseudo-cut based heuristics

In this section we review heuristics that use pseudo-cuts in order to cut-off portions of a solution space already examined in the previous solution process. A pseudo-cut consists of a 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 (Glover 2005). In addition, the heuristics reviewed in this section are based on solving series of reduced problems deduced from the original one. The idea is that the reduced MIP can be solved more quickly and if it is defined appropriately, the resulting solution will be a high-quality solution to the original MIP. Some earliest approaches for generating and exploiting small sub-problems are proposed in Glover and Laguna (1997), Glover (1977b), Soyster et al. (1978).

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

$$(MIP(J)) \begin{cases} maximize\ v = cx \\ s.t. \qquad Ax \leq b \\ \qquad\quad 0 \leq x_j \leq U_j,\ j \in N \\ \qquad\quad x_j \in [0,1], \quad j \in (\mathcal{I} - J) \\ \qquad\quad x_j \in \{0,1\}, \quad j \in J \end{cases} \tag{6}$$

The problem *reduced* from the original problem P and associated with x^0 and a subset $J \subseteq \mathcal{I}$ such that $x_j^0 \in \{0,1\}$ for all $j \in J$, is defined as:

$$P(x^0, J)\ \max\{cx : x \in X, x_j = x_j^0 \text{ for } j \in J\}. \tag{7}$$

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 $\delta(x, \widetilde{x})$:

$$MIP(P, \tilde{x}) \qquad \min\{\delta(\tilde{x}, x) : x \in X\}. \tag{8}$$

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

If C is a set of constraints, we will denote by $(P \mid C)$ the problem obtained by adding all constraints in C to the problem P. The set C in this section will represent the set of pseudo-cuts.

3.1 Local branching heuristics

Fischetti and Lodi (2003) proposed a *Local Branching* (LB) heuristic for 0–1 MIP based on soft variable fixing and the observation that the neighborhood of a feasible MIP–solution often contains solutions of possibly better quality. 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 Fischetti and Lodi (2008) that it could be used as a heuristic for building a first feasible solution.

The Local Branching heuristic performs soft fixing which requires that a certain number of variables take the same values as in the incumbent solution, without fixing any of those variables. More precisely, LB adds 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 within 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 the 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. A new problem is generated by reversing the constraint that defines the k-opt neighborhood of $\tilde{x}$ (i.e., explored part of the solution space is excluded) and adding a 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 the 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($k_0$);
1  Find an initial solution $\tilde{x}$ by CPLEX;
2  Set $k = k_0$;
3  Set $Y = X$ ;
4  repeat
5      if CPLEX finds $\tilde{x}' \in Y \cap \mathcal{N}^k(\tilde{x})$ better than $\tilde{x}$ then
6          $Y = Y - \mathcal{N}^k(\tilde{x})$;
7          set $\tilde{x} = \tilde{x}'$;
       else
8          if $Y \cap \mathcal{N}^k(\tilde{x}) \neq \emptyset$ then
9              $k = \lfloor k/2 \rfloor$;
           else
10             Set $k = k + \lfloor k/2 \rfloor$;
11             $Y = \{x \in X : \delta(x, \tilde{x}) \geq 1\}$;
           end
       end
   until Stopping criterion is satisfied;
   return $\tilde{x}$;
```

Hansen et al. (2006) proposed a 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 compared to the LB heuristic is the fact that it performs more systematic neighborhood exploration than Local Branching.

3.2 Iterative heuristics based on relaxations and pseudo-cuts

Hanafi and Wilbaut (2011) and Wilbaut and Hanafi (2009), 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 can be found. Unfortunately, in practice all of these heuristics turned out to be very slow and therefore the authors used them as heuristics by limiting the number of iterations permitted for their execution.

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 (Hanafi and Wilbaut 2011; Wilbaut and Hanafi 2009):

- *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 values 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 the LPA algorithm by solving a 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 the 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 a MIP relaxation based on the solution $\bar{x}$. In the next step, two reduced problems $P(\bar{x})$ and $P(\tilde{x})$ are solved and therefore two pseudo-cuts are added to the problem Q. The whole process is repeated for a 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.

3.3 Hybrid Variable Neighborhood decomposition search heuristics

Lazić et al. (2010) 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 by solving the LP-relaxation of the original problem to obtain 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. These distance values serve as criteria for choosing variables that will be fixed. Namely, at each iteration k variables whose indices correspond to the indices of k smallest δ_j values are fixed 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 (see Algorithm 7) is launched to conduct a 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 $y =$ `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 $y =$ `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 best solution found x^*.

Algorithm 7: Variable Neighborhood Descent branching

```
Function VNDS($P, t_{vnd}, t_{mip}, r_{max}, x'$);
1  Set $r = 1, t_{start} = CpuTime(), t = 0$;
2  Set $Q = P$;
3  while $t < t_{vnd}$ and $r \leq r_{max}$ do
4      set $time\_limit = min\{t_{mip}, t_{vnd} - t\}$;
5      $Q = (Q|\{\delta(x', x) \leq r\})$;
6      $x''$ = MIPsolve($Q, time\_limit, x'$);
7      switch solution status do
8          case OptSolFound:
9              Reverse last pseudo-cut into $\delta(x', x) > r + 1$;
10             $x' = x'', r = 1$;
11         case feasibleSolFound:
12             Replace last pseudo-cut with $\delta(x', x) \geq 1$;
13             $x' = x'', r = 1$;
14         case ProvenInfeasible:
15             Reverse last pseudo-cut into $\delta(x', x) > r + 1$;
16             $r = r + 1$;
17         case nofeasiblesolfound:
18             return $x''$;
       endsw
19     set $t_{end} = CpuTime(), t = t_{end} - t_{begin}$;
   end
20 return $x''$;
```

Hanafi et al. (2010) proposed a hybrid variable neighborhood decomposition search heuristic that constitutes an improved version of the variable neighborhood decomposition search heuristic proposed in Lazić et al. (2010). 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.

3.4 Summary

From the description of heuristics presented in this section we may conclude that their common features are: solve a reduced problem and add a pseudo-cut at each iteration of the solution process. In addition, all of them use a MIP solver to solve a reduced problem. However, the difference among them comes mainly from the way the reduced problem is defined. Local branching heuristics use the branching constraint to define the reduced problem while Iterative Heuristics Based on Relaxations and Pseudo Cuts as well as Variable Neighborhood Decomposition Search based heuristics define the reduced problem fixing some variables. In particular, Local branching heuristics use soft variable fixing to define the reduced problem, while the others are based on hard variable fixing. The hard variable fixing in Iterative Heuristics Based on Relaxations and Pseudo Cuts and Variable Neighborhood Decomposition Search based heuristics is accomplished choosing variables to be fixed according to the values they receive in the solution of a relaxed problem. As shown in Wilbaut et al. (2009), Iterative Heuristics Based on Relaxations and Pseudo Cuts turned out to be very

Algorithm 8: Variable neighborhood decomposition search based heuristic

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

efficient in solving the 0–1 multidimensional knapsack problem. Among them the worst performance was exhibited by MIPA heuristic, while the best one turned out to be IRH heuristic. According to results reported in Lazić et al. (2010), Hanafi et al. (2010) on instances from MIP library, it turns out that variable neighborhood decomposition search based heuristics outperform Local branching heuristics presented in this section, as well as relaxation induced neighbourhood search heuristic which will be described in Sect. 5. So, it appears that variable neighborhood decomposition approaches are very powerful and their application to other classes of problems may represent promising future research direction. Note that Local branching heuristic is available in SCIP non-commercial solver (Achterberg 2009) and CPLEX. All heuristics presented in this section can be applied in solving general MIP problems.

4 Pump and diving heuristics

This section is devoted to heuristics whose main purpose is to provide a first feasible solution for a MIP problem at hand. Here, they will be presented in the context of 0–1 MIP problems, although with small modifications they may be used to find a first feasible solution for a general MIP.

4.1 Feasibility pump

The Feasibility Pump (FP) heuristic (Algorithm 9) was proposed by Fischetti et al. (2005) which 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.,:

$$\min\{\delta(x, \tilde{x}) : Ax \leq b, 0 \leq x_j \leq U_j, j \in N\}. \tag{9}$$

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 particular sequence of points $\overline{x}$ and $\tilde{x}$ is visited again and again. That issue is resolved applying a random perturbation move to the current solution $\tilde{x}$ as soon as a cycle is detected. In the original implementation, this is performed by 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 selected 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($P$, $T$);
1 Solve the LP relaxation of $P$ to obtain an optimal LP basic solution $\overline{x}$;
2 repeat
3     $\tilde{x} = near(\overline{x})$;
4     Solve the $LP(P, \tilde{x})$ problem to obtain an optimal solution $\overline{x}$;
5     if cycle detected then
6         choose a random number $t \in [T/2, 3T/2]$;
7         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 approach was extended to the general feasibility pump, a heuristic for general mixed-integer problems (Bertacco et al. 2007). On the one hand, 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, the so called *Objective feasibility pump* was proposed in Achterberg and Berthold (2007). 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 Achterberg and Berthold (2007) indicate that this approach usually yields considerably higher-quality solutions than the basic FP. However, it generally requires much longer computational time.

4.2 Variable neighborhood pump

Hanafi et al. (2010) 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) (Hansen et al. 2006) and Feasibility pump heuristics (Fischetti et al. 2005). 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 one iteration of the FP pumping cycle is executed in order to obtain a near-feasible vector $\tilde{x}$. Then, a variable neighbourhood branching, adapted for 0–1 MIP feasibility (Hanafi et al. 2010), is applied on the solution $\tilde{x}$, 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($P$);
1  Set $proceed1$ = true;
2  while $proceed1$ do
3      Solve the LP relaxation of $P$ to obtain an optimal LP basic solution $\overline{x}$;
4      Set $\tilde{x} = near(\overline{x})$;
5      Set $proceed2$ = true;
6      while $proceed2$ do
7          if $\overline{x}$ is integer then return $\overline{x}$;
8          Solve the $LP(P, \tilde{x})$ problem to obtain an optimal solution $\overline{x}$;
9          if $\tilde{x} \neq near(\overline{x})$ then
               $\tilde{x} = near(\overline{x})$;
           else
10             Set $proceed2$ = false;
           end
       end
11     $k_{min} = \lfloor \delta(\tilde{x}, \overline{x}) \rfloor$; $k_{max} = \lfloor (|\mathcal{I}| - k_{min})/2 \rfloor$; $k_{step} = (k_{max} - k_{min})/5$;
12     $x' = \text{VNB}(P, \tilde{x}, k_{min}, k_{step}, k_{max})$;
13     if $x' = \tilde{x}$ then
14         $P = (P \mid \delta(x, \overline{x}) \geq k_{min})$; Update $proceed1$;
       else
15         return $x'$;
       end
   end
16 Output message: "No feasible solution found"; return $\tilde{x}$;
```

4.3 Diving heuristics

Lazić et al. (2014) 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 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 $\delta_1 \leq \delta_2 \leq \cdots \leq \delta_{|\mathcal{I}|}$. Then, VN diving successively solves 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 by the time the stopping criteria are met, the algorithm reports failure and returns the last (infeasible) integer solution.

The pseudo-code of the VN diving heuristic is given in 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 $y = \texttt{FindFirstFeasible}(P, t)$ 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.

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). 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), the 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 (8)). The variables to be fixed in the current subproblem are chosen from 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

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

```
Function VNdiving($P, d$);
1  Set $proceed1$ = true, $proceed2$ = true; Set timeLimit for subproblems;
2  while $proceed1$ 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 $\delta_j \leq \delta_{j+1}$, $j = 1, \dots, |\mathcal{I}| - 1$;
7      Set $n_d =\mid \{j \in \mathcal{I} : \delta_j \neq 0\} \mid$, $k_{step} = near(n_d/d)$, $k = |\mathcal{I}| - k_{step}$;
8      while $proceed2$ and $k \geq 0$ do
9          $J_k = \{1, \dots, k\}$; $x' =$ 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 $proceed2$;
       end
15     Update $proceed1$;
   end
16 Output message: "No feasible solution found"; return $\tilde{x}$;
```

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 integer reference 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 Lazić et al. 2014).

4.4 Summary

The differences among heuristics presented in this section may be summarized as follows: Only diving heuristics use hard variable fixing; VNP and diving heuristics use pseudo-cuts to reduce the solution space at each iteration, cutting of already examined portions (unlike to feasibility pump heuristics); VNP uses VNB as a subroutine which is based on soft variable fixing. According to results presented in Lazić et al. (2014), the best heuristics, among those presented in this section, are VN and SN diving heuristics. This fact may lead to a conclusion that hard variable fixing in combination with pseudo-cuts is the most suitable approach for getting high quality first feasible solution in short time for MIP problems. Thus, it would be interesting to examine

Algorithm 12: 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 $\widetilde{x}^0 = near(\overline{x})$;
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 $\alpha$;
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($P(\widetilde{x}^i, J_k)$, $timeLimit$);
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(MIP($Q_{i+1}$, $[\overline{x}]$), $timeLimit$);
19     if MIP($Q_{i+1}$, $near(\overline{x})$) is proven infeasible then Output message: "Problem $P$ is proven
       infeasible"; return;
20     $i = i + 1$;
   end
```

the behaviour of such approaches when applied on other classes of problems such as e.g., 0–1 MINLP. Note that Feasibility pump and Objective feasibility pump have been integrated in CPLEX, XPRESS and GUROBI commercial MIP solvers as well as in SCIP non-commercial MIP solver. All heuristics presented in this section may be applied to general MIP problems as well.

It is worth mentioning that there are also other heuristics for creating a first feasible that are usually used to solve certain problem classes. Some representatives are Fix-and-Relax, Insert-and-Fix, Fractional Relax-and-Fix heuristics that are usually applied to solve the Lot-sizing Problems see e.g., Pochet and Wolsey (2006), Stadtler (2003), Belvaux and Wolsey (2000), Toledo et al. (2015). Although they may be applied to any 0–1 MIP problem, these heuristics are not reviewed here since their merit have not been yet assessed in such context. So, applying these heuristics alone, or in combination with existing ones to solve general 0–1 MIP problems may be valuable future research direction.

5 Proximity heuristics

This section is devoted to heuristics that look for a high quality MIP feasible solution explored in the proximity of a solution at hand. As will be shown, the solution whose neighborhood is explored may be just LP feasible.

5.1 Ceiling heuristic

Saltzman and Hillier (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, a 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 a 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 let E_i, be the set of extreme directions emanating from $\bar{x}$ which lie on the i-th constraint hyperplane, i.e., the hyperplane that is the boundary of the half space $\sum_{j \in N} a_{ij} x_j \leq b_i$. 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:

$$\tilde{x}_j = \begin{cases} \lfloor x_j \rfloor, \text{ if } a_{i^*j} > 0 \\ \lfloor x_j + 0.5 \rfloor, \text{ if } a_{i^*j} = 0 \\ \lceil x_j \rceil, \text{ if } a_{i^*j} < 0 \end{cases} \tag{10}$$

- *Phase III (Improvement phase).* Once a feasible integer solution is found in Phase II, the 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, is capable of locating a 1-ceiling point (if one exists). 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 of this form exists).
  - 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 to 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.

5.2 Relaxation enforced neighbourhood search

The *Relaxation Enforced Neighborhood Search* heuristic (RENS) was proposed by Berthold (2007) as a new start heuristic for general MIPs working in the spirit of a large neighborhood search. RENS starts by solving an LP relaxation of the problem. After that all integer variables that received integer values in the solution of the LP relaxation are fixed while a large neighborhood search (LNS) is performed on remaining variables. The LNS is implemented by solving a 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.

5.3 Relaxation induced neighbourhood search

The *Relaxation Induced Neighborhood Search* (RINS), is an improving heuristic proposed by Danna et al. (2005). 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 likely be 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 for such variables RINS explores the neighborhood structure defined by the incumbent solution $\tilde{x}$ and LP-relaxation solution $\bar{x}$ as:

$$\mathcal{RIN}(\tilde{x}, \bar{x}) = \{x : x_j = \tilde{x}_j \text{ for } j \in \mathcal{I} \text{ such that } \bar{x}_j = \tilde{x}_j;\, x \text{ MIP feasible}\} \qquad (11)$$

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 exactly may require substantial time and thus it is preferable to solve sub-MIP approximatively, imposing the node limit, or alternatively 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 some subset of the nodes in the tree in order to find high quality solutions of the initial MIP within the imposed time limit.

5.4 Proximity search heuristics

Fischetti and Monaci (2014) 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 a defined MIP, is further improved by solving the initial problem fixing values of all binary variables to their values in $\tilde{x}$. The procedure finishes its work when specified (predefined) stopping conditions are 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 $\theta$;
  until Stopping criterion is satisfied;
  return $x^*$;
```

5.5 Summary

All heuristics presented in this section may be applied in solving general MIP problems except Proximity search heuristic. Among them RINS and Proximity search heuristics require a MIP feasible solution at the input, while the other two heuristics are able to construct MIP feasible solutions by themselves. In particular, as already pointed out RENS heuristic ia actually conceived as a heuristic for finding a first feasible solution for general MIPs. In addition, the above heuristics use different ways to define and explore the proximity of an incumbent solution. Ceiling heuristic tries to improve a feasible MIP solution simultaneously altering at most two of its components; RINS and RENS explore a neighborhood defined by fixing certain variables that receive integer values in the solution of LP relaxation; while Proximity search heuristics look for an improving solution which is at the same time closest to the incumbent one.
Among these heuristics RINS heuristic is implemented in CPLEX and Gurobi MIP solvers. The main purpose of RINS is to improve the best solution found so far. However, as already mentioned, since it is time consuming heuristic it is invoked only at the predefined number of nodes of branch-and-bound tree. On the other hand, SCIP solver contains RINS, RENS and proximity search heuristics.

6 Advanced heuristics

In this section we present heuristics that may be considered as frameworks for building new heuristics for 0–1 MIP. Therefore, we call these heuristics “Advanced heuristics”.

6.1 Parametric tabu search

The *parametric tabu search* (PTS), proposed by Glover (2006), is a general framework for building heuristics for solving general MIP problems. The main idea of PTS is to solve a 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 (Glover 1978), 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 Glover (2006), Sacchi and Armentano (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 a Branch-and-Bound procedure, these conditions are not imposed explicitly, but by indirectly incorporating them in the objective function. In that way the following LP problem is defined:

$$(LP(x', v^*)) \begin{cases} maximize\ cx + \sum_{j \in N^1(x')} c'_j(1 - x_j) + \sum_{j \in N^0(x')} c'_j x_j \\ s.t. \qquad Ax \leq b \\ \qquad\quad 0 \leq x_j \leq U_j,\ \ j \in N \\ \qquad\quad cx \geq (1 + \theta)v^* \end{cases} \tag{12}$$

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 the 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., a variable for which:

$$\begin{cases} x''_j < 1 \text{ for } j \in N^1(x') \text{ or} \\ x''_j > 0 \ \text{ for } j \in N^0(x'). \end{cases} \tag{13}$$

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 14. 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 goal infeasible or integer infeasible. If the solution x'' is goal infeasible, sets G_p and G_s are created by choosing their elements with reference to the 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 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 $x' \in ]0, 1[^n$;
2  $v^* = -\infty$;
3  repeat
4      if $LP(x', v^*)$ infeasible then break;
5      $x'' \leftarrow$ optimal solution of $LP(x', v^*)$;
6      if $x''$ MIP feasible then
7          $v^* \leftarrow cx''$;
8          $x^* \leftarrow x''$;
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 Glover (2006).

6.2 Metaheuristic search with inequalities and target objectives

Many adaptive memory and evolutionary metaheuristics for mixed integer programming include proposals for introducing inequalities and target objectives 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 approaches may be used in both intensification and diversification phases of a solution process. Glover and Hanafi (2010a, b) proposed enhanced versions of these approaches by introducing new inequalities that dominate those previously considered and new target objectives that underlie the creation of both inequalities and trial solutions. More precisely, they proposed the use of partial vectors and more general target objectives within inequalities in target solution strategies. The resulting procedures generate target objectives and solutions by exploiting proximity in the original space or the projected space. Additionally, they introduced more advanced approaches for generating the target objective based on exploiting mutually reinforcing notions of reaction and resistance.

The proximity procedure (Algorithm 16) for solving pure 0–1 MIP problems ($\mathcal{I} = N$), proposed in Glover and Hanafi (2010a, b), at each iteration solves the linear program defined as:

$$(LP(x', c', v^*)) \begin{cases} minimize \ \delta(c', x', x) = \sum_{j=1}^{n} c'_j (x_j(1 - x'_j) + x'_j(1 - x_j)) \\ s.t. \qquad Ax \leq b \\ \qquad\qquad 0 \leq x_j \leq U_j, \ \ j \in N \\ \qquad\qquad cx \geq (1 + \theta)v^* \end{cases} \tag{14}$$

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 by 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 procedure given in Algorithm 15 (see Glover and Hanafi 2010a for more details).

The rationale for using such a procedure is that the 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 a solution that differs 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 Glover and Hanafi (2010a) the suggested value for parameter $BaseCost$, of procedure for generating vector c', is 20.

Algorithm 15: Procedure for generating vector c'

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

Algorithm 16: Proximity procedure for 0–1 MIP

```
Function PS($BaseCost$);
1  $c' = c$;
2  $x' = 0$;
3  $v^* = -\infty$;
4  repeat
5      if $LP(x', c', v^*)$ infeasible then break;
6      $x'' \leftarrow$ optimal solution of $LP(x', c', v^*)$;
7      if $x''$ integer feasible then
8          $x^* \leftarrow x''$; //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 $j \in N$; //Construct the target solution $x'$ derived from $x''$
12         $c' \leftarrow$ Generate_vector ($BaseCost, x', x''$);
       end
   until Stopping criterion is satisfied;
   return $x^*$;
```

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.

The described proximity procedure, may be easily enhanced by updating the problem inequalities (adding and dropping constraints) in the way described in Glover and Hanafi (2010a). Further, in order to avoid a big difference between the components of two vectors c', used in two consecutive iterations, it is preferable 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 using a 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 by introducing advanced approaches for generating the target objective based on exploiting the mutually reinforcing notions of reaction and resistance as proposed in Glover and Hanafi (2010b).

6.3 OCTANE heuristic

Balas et al. (2001) 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 in 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. To describe the Ray Shooting Algorithm, we introduce the following definitions.

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 Berthold 2006). After that the first reachable facet is deduced according to the following theorem:

Theorem 2 *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. (2001), 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. (2001) proved the following statement.

Theorem 3 *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 a ray leads into the interior 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* (Berthold 2006). 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 provides a fruitful direction for finding feasible points.

6.4 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 Glover (1995a) 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 (1997) elaborated these ideas and extended them to General Mixed Integer Programs by means of a more general definition of directional rounding.

Building on above ideas, Glover et al. (2000) 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.

6.5 Restrict-and-relax search

Restrict-and-Relax search (Guzelsoy et al. 2013) is a branch-and-bound based algorithm that dynamically changes sets of fixed and non-fixed (free) variables. At the beginning, it defines a restricted 0–1 MIP, by fixing some variables, from where it starts a branch-and-bound based exploration of the solution space. At any node of the search tree, it may selectively relax previously fixed variables, fix (restrict) additional variables, or relax and fix variables at the same time. Since Restrict-and-Relax search operates with restricted 0–1 MIP in a dynamic way it is very suitable for solving instances that are too large to be fully loaded into memory. In addition, Restrict-and-Relax search is able to find high-quality feasible solutions more quickly than a traditional search as well as to prove optimality.

6.6 Summary

The presented heuristics may be seen as frameworks for generating heuristics for 0–1 MIP problems. Namely, for each ingredient of these heuristics there is more than one option how it may be implemented. Therefore choices of different ingredients and their way of implementation obviously lead to different heuristics. So, in our point of view generating different heuristics from these frameworks may represent very promising research directions. However, to the best of our knowledge only OCTANE, restrict-and-relax and the core parametric tabu search heuristics have been implemented and tested on 0–1 MIP instances. According to results reported in Balas et al. (2001), OCTANE heuristic is highly competitive with Pivot and Shift heuristic (see Sect. 2). Restrict-and-relax outperforms SYMPHONY (Ralphs and Güzelsoy 2005). On the other hand as shown in Sacchi and Armentano 2011, the core parametric tabu search is highly competitive with Feasibility Pump and Objective Feasibility pump (see Sect. 4). All these facts imply that there is still work to do in this area which may lead to powerful heuristics for 0–1 MIP. In addition, Parametric Tabu search and Star path with directional rounding may be also applied to general mixed integer programs. So, developing heuristics from these frameworks and assessing their performance on benchmark instances for general MIPs represent another research line. Note that OCTANE heuristic is integrated in SCIP solver.

Table 1 Classification of heuristics

Type of heuristic		
Constructive	Improvement	Constructive & Improvement
FP heuristics	Local branching	Pivot and complement
VNP	VN branching	Pivot and shift
VN diving	RINS	Pivot and tabu
SN diving	Proximity search	Pivot cut and dive
RENS	HVNDS heuristics	Iterative heuristics (Sect. 3.2)
		Ceiling heuristic
		PTS
		Metaheuristic search (Sect. 6.2)
		OCTANE
		Star path with directional rounding
		Restrict-and-relax

7 Classification and summary of main components of MIP heuristics

It is not easy to classify the heuristics based on mathematical programming techniques, since there are different ways to classify heuristic algorithms. In this survey we have proposed an implicit classification induced by the sections proposed in this paper. Despite that some heuristics can appear in more than one section. However, these heuristics can be classified in two very general classes: constructive heuristics and improvement heuristics. Constructive heuristics start from scratch and proceed through a set of steps, to produce a solution without guaranty that the generated solution is feasible. Improvement heuristics start with a solution (feasible or infeasible) and iteratively execute improving steps to find high quality solutions. In general, successful heuristics employee a two-phase approach: a constructive heuristic generates an initial solution and an improvement heuristic tries to produce better solution than the initial one. Therefore, Table 1 presents the classification of surveyed heuristics into the following three groups: constructive, improvement and constructive&improvement heuristics.

Moreover in this section, we identify the main components shared by heuristics based on mathematical programming. As main components of the heuristics presented above we may identify the following:

– *Move operators*. The typical move operators are shifting, pivoting and rounding. Note that in the context of 0–1 MIP problems shifting operators reduce to complementing operators.
– *Variable fixing*. Soft and hard variable fixing are standard fixing techniques. As shown, hard variable fixing is accomplished assigning certain values to the subset of variables, while soft variable fixing requires that certain number of variables take the same values as in the incumbent solution, without fixing explicitly any of those variables.

Table 2 Overview of Components used by heuristics

Heuristic	Operators			Variable fixing		Pseudo cuts	Decomposition	Neighborhood		Proximity obj.	MIP solver
	Shifting	Rounding	Pivoting	Hard	Soft			Single	Multiple		
Pivot and Complement	+	+	+	+	–	–	–	–	+	–	–
Pivot and Shift	+	–	+	–	+	–	–	–	+	–	+
Pivot and Tabu	–	–	+	–	–	–	–	+	–	–	–
Pivot Cut and Dive	–	–	+	–	–	+	–	–	+	-	-
Local Branching	–	–	–	–	+	+	–	+	–	–	+
Variable Neighborhood Branching	–	–	–	–	+	+	–	–	+	-	+
Iterative heuristics (Section 3.2)	–	–	–	+	–	+	–	–	–	–	+
HVNDS heuristics	–	–	–	+	+	+	+	–	+	–	+
FP heuristics	–	+	–	–	–	–	–	–	–	+	+
VNP	–	–	–	–	+	+	–	–	+	+	+
VN diving	–	+	–	+	–	+	+	–	+	–	+
SN diving	–	+	–	+	–	+	–	+	–	+	+
Ceiling heuristic	+	+	–	–	–	–	–	–	–	–	–
RENS	–	–	–	+	–	–	–	+	–	–	+
RINS	–	–	–	+	–	–	–	+	–	–	+
Proximity search	–	–	–	–	–	–	–	–	–	+	+
PTS	–	–	–	–	–	–	–	–	–	–	–
Metaheuristic Search (Sect. 6.2)	–	+	–	–	–	–	–	–	–	+	–
OCTANE	–	–	–	–	–	–	–	–	–	–	–
Star Path with directional rounding	–	–	–	–	–	–	–	–	–	–	–
Restrict-and-Relax	–	–	–	+	–	–	–	–	–	–	–

- *Pseudo cuts*. Excluding portion from the space search is the main purpose of the pseudo-cuts, in order to cut-off the solutions already visited during the process.
- *Problem decomposition*. Problem decomposition refers here whether variable neighborhood decomposition search (VNDS) principles are used or not.
- *Neighborhood structure examined*. We distinguish two very general types of neighborhood structures: single (only one neighborhood structure is examined throughout solution process) and multiple (multiple neighborhood structures are examined throughout solution process)
- *Proximity objective function*. The proximity objective function is the one defined as the distance of a solution from the incumbent solution.
- *MIP solver*. Heuristics may rely on exact MIP solver which is usually used to solve reduced MIP problems deduced from the initial one.

In Table 2, we present which of above components are used by certain heuristic. Sign '+' in the table means that certain ingredient is used, while sign '−' stands for the opposite.

8 Concluding remarks

This paper provides a survey of heuristics based on mathematical programming for 0–1 mixed integer programs without exploiting any special structure of the problem at hand. In addition to describing the main ideas of these heuristics we provide pseudo-codes for the key methods using a uniform coding template. Some of presented heuristics are concerned with finding a first MIP feasible solution which is generally NP hard problem by itself. Among the better known heuristics of this type are the Feasibility Pump heuristics, the Variable Neighborhood Pump approach, and the Single and Variable Neighborhood Diving heuristics. The second class consists of improvement procedures for 0–1 MIP that require a feasible MIP solution as the input. Heuristics of this type include Local Branching, Variable Neighborhood Branching, and Relaxation Induced Neighborhood Search. The last class of heuristics are those that neither require a feasible MIP solution as the input nor finish their work after finding the first feasible solution. Prominent members of this group include Pivot and Complement, Pivot and Shift, Tabu Search, Pivot-Cut-and-Dive, variable neighborhood decomposition based heuristics, iterative heuristics based on relaxations and advanced heuristics. The best heuristics for finding a first feasible solution are those based on diving: Variable Neighborhood and Single Neighborhood diving heuristics. On the other hand, as the best improvement heuristics may be identified those based on variable neighborhood decomposition search (VNDS). This may be explained by the fact that VNDS based heuristics operate with series of small sub-problems of the original problem unlike the other heuristics which operate mainly with the original problem even if it is large scale problem. Most of these heuristics are already embedded in some commercial or non-commercial solvers such as CPLEX, GUROBI, GAMS, XPRESS, SYMPHONY, COIN, etc. Note that most of existing software do not provide the functionality to implement surveyed heuristics easily. For example, restrict-and-relax search cannot be implemented in the commercial solvers CPLEX, Gurobi, and XPRESS since they do not provide capability to start from a restricted problem and to fix and unfix variables at selected nodes in the search tree. However, we believe that if the solver allows more flexibility to the user, this may lead in solving larger and harder problems that are elusive for the solver for the moment. Therefore potentially valuable directions for future research consist in developing new advanced procedures that combine the ideas of both heuristic and exact approaches. Some initiatives for achieving this are already underway, for example the combination of variable neighborhood decomposition search heuristic with iterative linear programming heuristic.

Further, we can observe that the most of advanced heuristics are at the starting point regarding the evaluation, implementation and analysis of their merit. For example, to the best of our knowledge there is only one basic implementation of parametric tabu search, while there is neither implementation of the Metaheuristic Search with Inequalities and Target Objectives nor of Star Path with directional rounding. Thus, future research direction may also include implementing and testing new 0–1 MIP heuristics that stem from these frameworks.

Acknowledgements This work is supported by the ELSAT2020 project which is co-financed by the European Union with the European Regional Development Fund, the French state and the Hauts de France Region Council. We are indebted to Fred Glover for his help in improving English of this paper, and also owe our gratitude to reviewers whose comments have helped to improve the paper's exposition.

References

Aboudi, R., Jörnsten, K.: Tabu search for general zero-one integer programs using the pivot and complement heuristic. ORSA J. Comput. **6**(1), 82–93 (1994)

Aboudi, R., Hallefjord, A., Helming, R., Jornsten, K.: A note on the pivot and complement heuristic for 0–1 programming problems. Oper. Res. Lett. **8**, 21–23 (1989)

Achterberg, T., Berthold, T.: Improving the feasibility pump. Discrete Optim. **4**(1), 77–86 (2007)

Achterberg, T.: Scip: solving constraint integer programs. Math. Program. Comput. **1**(1), 1–41 (2009)

Balas, E., Martin, C.: Pivot and complement-a heuristic for 0–1 programming. Manag. Sci. **26**, 86–96 (1980)

Balas, E., Ceria, S., Dawande, M., Margot, F., Pataki, G.: Octane: a new heuristic for pure 0–1 programs. Oper. Res. **49**, 207–225 (2001)

Balas, E., Schmieta, S., Wallace, C.: Pivot and shift-a mixed integer programming heuristic. Discrete Optim. **1**, 3–12 (2004)

Ball, M.O.: Heuristics based on mathematical programming. Surv. Oper. Res. Manag. Sci. **16**(1), 21–38 (2011)

Beasley, J.E.: Lagrangean heuristics for location problems. Eur. J. Oper. Res. **65**(3), 383–399 (1993)

Belvaux, G., Wolsey, L.A.: Bcprod: a specialized branch-and-cut system for lot-sizing problems. Manag. Sci. **46**(5), 724–738 (2000)

Bertacco, L., Fischetti, M., Lodi, A.: A feasibility pump heuristic for general mixed-integer problems. Discrete Optim. **4**(1), 63–76 (2007)

Berthold, T.: Primal heuristics for mixed integer programs. Master's thesis, Technischen Universitat Berlin (2006)

Berthold, T.: Rens-relaxation enforced neighborhood search. Technical report tr-07-28, ZIB, Berlin (2007)

Berthold, T.: Heuristic algorithms in global minlp solvers. Ph.D. thesis, Technischen Universitat Berlin (2014)

Bonami, P., Kilinç, M., Linderoth, J.: Algorithms and software for convex mixed integer nonlinear programs. In: Lee, J., Leyffer, S. (eds.) Mixed Integer Nonlinear Programming, pp. 1–39. Springer, Berlin (2012)

Burer, S., Letchford, A.N.: Non-convex mixed-integer nonlinear programming: a survey. Surv. Oper. Res. Manag. Sci. **17**(2), 97–106 (2012)

Cabot, A.V., Hurter Jr., A.P.: An approach to zero-one integer programming. Oper. Res. **16**(6), 1206–1211 (1968)

Danna, E., Rothberg, E., Pape, L.C.: Exploring relaxation induced neighborhoods to improve mip solutions. Math. Program. **102**(1), 71–90 (2005)

Dantzig, G.: Programming in a Linear Structure. Comptroller. USAF, Washington, DC (1948)

Dantzig, G.: Linear Proxramming and Extensions. Princeton University Press, Princeton (1963)

Eckstein, J., Nediak, M.: Pivot, cut and dive: a heuristic for 0–1 mixed integer programming. J. Heuristics **13**, 471–503 (2007)

Farahani, R.Z., Hekmatfar, M., Arabani, A.B., Nikbakhsh, E.: Hub location problems: a review of models, classification, solution techniques, and applications. Comput. Ind. Eng. **64**(4), 1096–1109 (2013)

Fischetti, M., Lodi, A.: Local branching. Math. Program. **98**, 23–47 (2003)

Fischetti, M., Lodi, A.: Repairing mip infeasibility through local branching. Comput. Oper. Res. **35**, 1436–1445 (2008)

Fischetti, M., Lodi, A.: Heuristics in mixed integer programming. In: Cochran, J.J. (ed.) Wiley Encyclopedia of Operations Research and Management Science, vol. 8, pp. 738–747. Wiley, New York (2011)

Fischetti, M., Monaci, M.: Proximity search for 0–1 mixed-integer convex programming. J. Heuristics **20**(6), 709–731 (2014)

Fischetti, M., Glover, F., Lodi, A.: The feasibility pump. Math. Program. **104**(1), 91–104 (2005)

Glover, F.: A note on linear programming and integer infeasibility. Oper. Res. **16**, 1212–1216 (1968)

Glover, F.: Heuristics for integer programming using surrogate constraints. Decis. Sci. **8**, 156–166 (1977a)

Glover, F.: Heuristics for integer programming using surrogate constraints. Decis. Sci. **8**(1), 156–166 (1977b)

Glover, F.: Parametric branch and bound. OMEGA, Int. J. Manag. Sci. **6**, 1–9 (1978)

Glover, F.: Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res. **13**(5), 533–549 (1986)

Glover, F.: Scatter search and star-paths: beyond the genetic metaphor. OR Spektrum **17**, 125–137 (1995a)

Glover, F.: Tabu thresholding: Improved search by nonmonotonic trajectories. ORSA J. Comput. **7**, 426–442 (1995b)

Glover, F.: Adaptive memory projection methods for integer programming. In: Metaheuristic Optimization Via Memory and Evolution, pp. 425–440. Springer (2005)

Glover, F.: Parametric tabu-search for mixed integer programs. Comput. Oper. Res. **33**, 2449–2494 (2006)

Glover, F., Laguna, M.: General purpose heuristics for integer programming-part ii. J. Heuristics **3**, 161–179 (1997)

Glover, F., Hanafi, S.: Metaheuristic search with inequalities and target objectives for mixed binary optimization-part i: exploiting proximity. Int. J. Appl. Metaheuristic Comput. (IJAMC) **1**, 1–15 (2010a)

Glover, F., Hanafi, S.: Metaheuristic search with inequalities and target objectives for mixed binary optimization-part ii: Exploiting reaction and resistance. Int. J. Appl. Metaheuristic Comput. (IJAMC) **1**(2), 1–17 (2010b)

Glover, F., Taillard, E., de Werra, D.: A user's guide to tabu search. Ann. Oper. Res. **41**, 3–28 (1993)

Glover, F., Løkketangen, A., Woodruff, D.L.: OR computing tools for modeling, optimization and simulation: interfaces in computer science and operations research. In: Laguna, M., Gonzlez-Velarde, J. (eds.) OR Computing Tools for Modeling, pp. 299–317. Kluwer Academic Publishers, Dordrecht (2000)

Guzelsoy, M., Nemhauser, G., Savelsbergh, M.: Restrict-and-relax search for 0–1 mixed-integer programs. EURO J. Comput. Optim. **1**(1–2), 201–218 (2013)

Hanafi, S., Wilbaut, C.: Improved convergent heuristics for the 0–1 multidimensional knapsack problem. Ann. OR **183**, 125–142 (2011)

Hanafi, S., Lazić, J., Mladenović, N.: Variable neighbourhood pump heuristic for 0–1 mixed integer programming feasibility. Electron. Notes Discrete Math. **36**, 759–766 (2010a)

Hanafi, S., Lazić, J., Mladenović, N., Wilbaut, C., Crévits, I.: Hybrid variable neighbourhood decomposition search for 0–1 mixed integer programming problem. Electron. Notes Discrete Math. **36**, 883–890 (2010b)

Hansen, P., Mladenović, N., Urosević, D.: Variable neighborhood search and local branching. Comput. Oper. Res. **33**(10), 3034–3045 (2006)

Hansen, P., Mladenović, N., Todosijević, R., Hanafi, S.: Variable neighborhood search: basics and variants. EURO J. Comput. Optim. (2016). doi:10.1007/s13675-016-0075-x

Holmberg, K., Yuan, D.: A lagrangian heuristic based branch-and-bound approach for the capacitated network design problem. Oper. Res. **48**(3), 461–481 (2000)

Imai, A., Nishimura, E., Current, J.: A lagrangian relaxation-based heuristic for the vehicle routing with full container load. Eur. J. Oper. Res. **176**(1), 87–105 (2007)

Jena, S.D., Cordeau, J.F., Gendron, B.: Lagrangian heuristics for large-scale dynamic facility location with generalized modular capacities. Tecnical report CIRRELT—2014–21 (2014)

Lazić, J., Hanafi, S., Mladenović, N., Urosević, D.: Variable neighbourhood decomposition search for 0–1 mixed integer programs. Comput. Oper. Res. **37**, 1055–1067 (2010)

Lazić, J., Todosijević, R., Hanafi, S., Mladenović, N.: Variable and single neighbourhood diving for MIP feasibility. Yugoslav J. Oper. Res. (2014). doi:10.2298/YJOR140417027L

Lee, J., Leyffer, S.: Mixed Integer Nonlinear Programming, vol. 154. Springer, Berlin (2011)

Løkketangen, A., Jörnsten, K., Storøy, S.: Tabu search within a pivot and complement framework. Int. Trans. Oper. Res. **1**(3), 305–316 (1994)

Lokketangen, A., Glover, F.: Solving zero-one mixed integer programming problems using tabu search. Eur. J. Oper. Res. **106**, 624–658 (1998)

Marsten, R.: XMP Technical Reference Manual. Dept. of Management Information Systems, University of Arizona, Tucson, AZ (1987)

Mönch, L., Fowler, J.W., Dauzère-Pérès, S., Mason, S.J., Rose, O.: A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. J. Sched. **14**(6), 583–599 (2011)

Pochet, Y., Wolsey, L.A.: Production Planning by Mixed Integer Programming. Springer Science & Business Media, New York (2006)

Ralphs, T.K., Güzelsoy, M.: The symphony callable library for mixed integer programming. In: Golden, B.L., Raghavan, S., Wasil, E.A. (eds.) The Next Wave in Computing, Optimization, and Decision Technologies, pp. 61–76. Springer, New York (2005)

Rego, C., Gamboa, D., Glover, F., Osterman, C.: Traveling salesman problem heuristics: leading methods, implementations and latest advances. Eur. J. Oper. Res. **211**(3), 427–441 (2011)

Sacchi, L.H., Armentano, V.A.: A computational study of parametric tabu search for 0–1 mixed integer programs. Comput. Oper. Res. **38**(2), 464–473 (2011)

Saltzman, R.M., Hillier, F.S.: A heuristic ceiling point algorithm for general integer linear programming. Manag. Sci. **38**(2), 263–283 (1992)

Soyster, A., Lev, B., Slivka, W.: Zero-one programming with many variables and few constraints. Eur. J. Oper. Res. **2**(3), 195–201 (1978)

Stadtler, H.: Multilevel lot sizing with setup times and multiple constrained resources: Internally rolling schedules with lot-sizing windows. Oper. Res. **51**(3), 487–502 (2003)

Toledo, F.M.B., Armentano, V.A.: A lagrangian-based heuristic for the capacitated lot-sizing problem in parallel machines. Eur. J. Oper. Res. **175**(2), 1070–1083 (2006)

Toledo, C.F.M., da Silva, Arantes M., Hossomi, M.Y.B., França, P.M., Akartunalı, K.: A relax-and-fix with fix-and-optimize heuristic applied to multi-level lot-sizing problems. J. Heuristics **21**(5), 687–717 (2015)

Vidal, T., Crainic, T.G., Gendreau, M., Prins, C.: Heuristics for multi-attribute vehicle routing problems: a survey and synthesis. Eur. J. Oper. Res. **231**(1), 1–21 (2013)

Wilbaut, C., Hanafi, S.: New convergent heuristics for 0–1 mixed integer programming. Eur. J. Oper. Res. **195**, 62–74 (2009)

Wilbaut, C., Salhi, S., Hanafi, S.: An iterative variable-based fixation heuristic for the 0–1 multidimensional knapsack problem. Eur. J. Oper. Res. **199**(2), 339–348 (2009)