Journal of Heuristics (2022) 28:329–350
https://doi.org/10.1007/s10732-022-09496-2

Weighted iterated local branching for mathematical programming problems with binary variables

Filipe Rodrigues[1] · **Agostinho Agra**[2] · **Lars Magnus Hvattum**[3] · **Cristina Requejo**[2]

Received: 15 March 2021 / Revised: 16 March 2022 / Accepted: 30 March 2022 /
Published online: 16 April 2022

Abstract

Local search algorithms are frequently used to handle complex optimization problems involving binary decision variables. One way of implementing a local search procedure is by using a mixed-integer programming solver to explore a neighborhood defined through a constraint that limits the number of binary variables whose values are allowed to change in a given iteration. Recognizing that not all variables are equally promising to change when searching for better neighboring solutions, we propose a weighted iterated local branching heuristic. This new procedure differs from similar existing methods since it considers groups of binary variables and associates with each group a limit on the number of variables that can change. The groups of variables are defined using weights that indicate the expected contribution of flipping the variables when trying to identify improving solutions in the current neighborhood. When the mixed-integer programming solver fails to identify an improving solution in a given iteration, the proposed heuristic may force the search into new regions of the search space by utilizing the group of variables that are least promising to flip. The weighted iterated local branching heuristic is tested on benchmark instances of the optimum satisfiability problem, and computational results show that the weighted method is superior to an alternative method without weights.

Keywords Neighborhood search · Mixed-integer programming · Matheuristic · Boolean optimization

✉ Lars Magnus Hvattum
hvattum@himolde.no

1 ISEG-School of Economics and Management, University of Lisbon, Lisbon, Portugal

2 Department of Mathematics, University of Aveiro, Portugal, Portugal

3 Faculty of Logistics, Molde University College, Norway, Norway

1 Introduction

For optimization problems that cannot be solved to optimality within a reasonable time, heuristic search strategies are of key importance to obtain high-quality solutions. Several types of heuristic strategies have been proposed in the literature (Blum and Roli 2003; Boussaïd et al. 2013), and one of the most popular is to search for better solutions by exploring neighborhoods of given reference solutions. A common goal in the design of local search based metaheuristics (Sörensen and Glover 2015) is to provide ways to escape from local optima. Examples can be found in simulated annealing (Kirkpatrick et al. 1983) that uses randomness to allow worsening moves to escape from local optima, guided local search (Alsheddy et al. 2016) that uses penalties in the move evaluation function, iterated local search (Stützle and Ruiz 2017) that includes a perturbation step to move away from a local optimum, and tabu search (Glover and Laguna 1997) that uses memory structures such as tabu lists to prevent the return to recently visited local optima.

Recently, researchers have to an increasing extent considered variations over this theme where the exploration of neighborhoods is in part performed by formulating a mixed-integer programming problem (MIP) which is then solved using available commercial solvers. This corresponds to an integrative combination of exact and heuristic algorithms (Raidl and Puchinger 2008), with the local search being the master algorithm and the MIP solver being the integrated slave. Common examples of such matheuristics include local branching (Fischetti and Lodi 2003), relaxation induced neighborhood search (Danna et al. 2005), proximity search (Fischetti and Monaci 2014), and other methods based on large-neighborhood search frameworks (Rothberg 2007).

When exploring large neighborhoods using MIP formulations, techniques to reduce the time spent exploring the neighborhoods are relevant. One such technique involves fixing variables. In Wang et al. (2011) the authors investigated strategies for fixing variables in a tabu search for binary quadratic programming problems. When solving a network design problem, (González et al. 2016) used a variable fixing heuristic to generate an initial solution and then a local branching strategy to improve the solution. In Sadykov et al. (2019), heuristic schemes involving different forms of variable fixing were explored within a branch-and-price algorithm. Reduced costs were applied in Sarayloo et al. (2021) to guide variable fixing to solve a stochastic network design problem, while (Dauer and de Athayde Prata 2021) investigated a MIP heuristic with variable fixing for a vehicle scheduling problem.

Typically, local search based matheuristics do not distinguish between variables when defining a neighborhood. For example, in local branching, constraints are used to define the current neighborhood by limiting the number of binary variables that can flip (i.e., change their value from 0 to 1 or from 1 to 0) relative to the current solution. However, these local branching constraints do not exploit information about which variables are more likely to be flipped when moving from the current solution to an improving neighboring solution. A related method called proximity search uses the Hamming distance in the objective function to guide the search. However, this function also does not provide guidance to distinguish between the quality of equally distant solutions, nor does it incorporate available information regarding the preferred values

of variables. In a recent paper (Rodrigues et al. 2021) a proximity search heuristic using different weights for individual variables was successfully applied to three different combinatorial optimization problems. This provided evidence that it can be useful, prior to exploring a neighborhood of solutions, to evaluate the possible consequences of flipping binary variables and weighting the variables accordingly when performing the neighborhood exploration.

This paper introduces a new matheuristic that controls and guides the search by introducing individual variable weights when building local branching constraints. We refer to this method as a weighted iterated local branching (WILB) heuristic, as it uses ideas from local branching in a framework where the search may continue beyond a local optimum.

The WILB focuses on a search that modifies the values of binary variables. However, the WILB is designed with the assumption that not all binary variables are equally promising to flip when searching for an improving solution. The search defines a partition of the binary variables into three groups and associates with each group an independent local branching constraint. This allows the search to reduce one of the drawbacks of local branching, which consists of tuning the parameter that controls the number of variables that are allowed to flip. If this parameter is set to a small value, the neighborhood becomes very small and the search will quickly be trapped in a local optimum. If the parameter is set to a large value, the resulting problem may become too hard to solve, and the search may be unable to find an improving solution within an acceptable time limit. Defining three local branching constraints, we can guide the search by better controlling the search space using the information available at each stage.

To define the partition, a weight is first associated with each binary variable by taking into account three factors. The first factor measures the contribution of the variable to the objective function value; the second measures the general contribution of the variable to feasibility; and the third measures the immediate consequences of flipping the variable on the structure of the current solution. Considering the weights, one group is formed by the variables deemed most promising to flip in order to improve the current solution without changing the structure of the solution significantly, another group is formed by the variables considered least promising to flip or that change the structure of the solution significantly, while the remaining variables form a separate group. Each local branching constraint has a different right-hand side value, defined according to the expected contribution of the associated group of variables for improving the current feasible solution and changing the structure of the current solution. Furthermore, the set of variables that are least promising to flip or change the structure of the solution the most can also guide the local search to different regions of the search space whenever the solution obtained by the MIP solver is not better than the incumbent solution.

The new matheuristic is tested on instances of the optimum satisfiability problem (OptSAT) (Davoine et al. 2003). There are two reasons for this choice. First, several of the benchmark instances in the literature were generated through a transformation of instances from other NP-hard problems. Thus, by testing these instances, we indirectly test the method also on other problems. Second, the OptSAT has a nice structure that permits a clear interpretation of the three defined weight factors, which allows us to conduct this research as a proof of concept.

The contributions of this paper are the following.

1. Propose the weighted iterated local branching heuristic, a new matheuristic that constructs distinct groups of binary variables and associates an independent local branching constraint to each of these groups.
2. Use the group of variables that are least promising to flip when searching for better solutions to also direct the local search into different regions of the search space after every iteration where the search has failed to identify an improving solution.
3. Present the benefits of using the WILB to find better solutions compared to those obtained by the unweighted local branching and by a commercial solver with an imposed time limit through computational experiments on instances of the OptSAT.

The paper is organized as follows. In Sect. 2, we review the concept of local branching as used iteratively in a local search heuristic and introduce the new WILB. In Sect. 3, we present the OptSAT used to test the proposed WILB. The computational experiments are reported in Sect. 4 and the main conclusions are drawn in Sect. 5.

2 Local search procedures

In this section we review a general heuristic based on local branching, which we refer to as local branching heuristic (LBH) and then introduce the proposed weighted iterated local branching (WILB) heuristic. In the following, $\mathbf{x}$ and $\mathbf{y}$ denote vectors of decision variables, whereas x_j and y_j denote the j^{th} component of the corresponding vector. Moreover, $\bar{\mathbf{x}}$ denotes the value of the decision variables $\mathbf{x}$ in the current iteration of the procedure.

Consider a general MIP problem of the form

$$\begin{aligned} \max \quad & f(\mathbf{x}, \mathbf{y}) \\ s.t. \quad & g_i(\mathbf{x}, \mathbf{y}) \leq 0, \qquad i \in I, \\ & \mathbf{x} \in \{0, 1\}^n, \\ & \mathbf{y} \in \mathbb{R}^p \times \mathbb{Z}^q, \end{aligned} \tag{2.1}$$

where I is a set of indices, and $f, g_i,\ i \in I$, are real functions defined over $\mathbb{R}^{n+p+q}$. Both the LBH and the WILB specifically focus on the binary variables labeled $\mathbf{x}$, with $N = \{1, ..., n\}$ being the set of indices of the binary variables $\mathbf{x}$. Although the computational experiments of this paper consider a problem with only binary variables, the following discussion is kept general by keeping the possibility of non-binary variables included in $\mathbf{y}$.

2.1 Local branching heuristic

Different versions of heuristics based on local branching have been used for solving complex MIP problems, for example in vehicle routing (Hernandez et al. 2019), network design (Rodríguez-Martín and Salazar-González 2010), and production scheduling (Samavati et al. 2017). The main idea of such procedures is to repeatedly

define a neighborhood around the current solution, and then explore this neighborhood to find an improved solution. One way of implementing a LBH is through a local branching constraint of the form

$$LB^{\Delta} := \sum_{j \in N \mid \bar{x}_j = 0} x_j + \sum_{j \in N \mid \bar{x}_j = 1} (1 - x_j) \leq \Delta. \tag{2.2}$$

Fischetti and Lodi (2003) proposed this local branching constraint which restricts the number of binary variables that may change their value relative to an incumbent solution. That is, the constraint induces a neighborhood of the incumbent solution, which is defined as the set of solutions that can differ from the incumbent in at most Δ of the x-variables. The local branching constraint was originally proposed together with an inverse constraint

$$ILB^{\Delta} := \sum_{j \in N \mid \bar{x}_j = 0} x_j + \sum_{j \in N \mid \bar{x}_j = 1} (1 - x_j) \geq \Delta + 1. \tag{2.3}$$

which in combination with constraint (2.2) partitions the solution space and allows a branching scheme. In terms of identifying primal solutions of high quality, it has been claimed that this inverse constraint is less effective in practice (Hill and Voß 2018), and when using the local branching idea heuristically, most authors focus only on repeatedly applying the LB^{Δ} constraint. Thus, while successful applications of local branching using ILB^{Δ} have also been reported (Hernandez et al. 2019; Rei et al. 2010), in this paper we focus on heuristics where only LB^{Δ} applies. A LBH using a local branching constraint is given in Algorithm 1.

Algorithm 1 Local Branching Heuristic

1: Let $(\mathbf{x}^0, \mathbf{y}^0)$ be an initial feasible solution and Δ_0 a given parameter
2: set $(\bar{\mathbf{x}}, \bar{\mathbf{y}}) := (\mathbf{x}^0, \mathbf{y}^0)$, $f^{Best} := f(\mathbf{x}^0, \mathbf{y}^0)$, and $\Delta := \Delta_0$
3: **repeat**
4: set $f^{Previous} := f(\bar{\mathbf{x}}, \bar{\mathbf{y}})$
5: start with the pure MIP model and add the local branching constraint LB^{Δ}
6: run the MIP solver to obtain a new solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$
7: **if** $f(\bar{\mathbf{x}}, \bar{\mathbf{y}}) > f^{Best}$ **then**
8: set $f^{Best} := f(\bar{\mathbf{x}}, \bar{\mathbf{y}})$
9: **end if**
10: **if** $f(\bar{\mathbf{x}}, \bar{\mathbf{y}}) > f^{Previous}$ **then**
11: $\Delta := \Delta_0$
12: go to Step 3
13: **else**
14: increase Δ and go to Step 3
15: **end if**
16: **until** a given stopping criterion is reached

The LBH procedure starts from an initial feasible solution for the problem and requires the initialization of the parameter Δ used in the LB^{Δ} constraint. Here, the initial solution is obtained by executing a MIP solver for a limited amount of time.

At each iteration (Steps 3–16), a local branching constraint LB^{Δ} is added to the model (Step 5) which is then given to a MIP solver that is run until a termination criterion is reached (Step 6), and a new incumbent (feasible) solution is obtained. If the objective function value of the obtained solution is better than the current best value, the best solution is updated (Step 8). When the objective function value of the new incumbent solution is no better than the objective function value of the solution found in the previous iteration, the value of Δ increases (leading to a larger neighborhood). Otherwise, the value of Δ is not changed. The algorithm repeats the process (Steps 3–16) until it reaches a pre-defined stopping criterion (Step 16).

Different descriptions of LBH procedures using local branching constraints can be found in the literature regarding how to perform the irregular iterations which occurs when no improvement is found within the current neighborhood (Step 14) (Faria et al. 2019). In this paper we follow a very simple strategy for such iterations that consists of increasing the value of Δ. We made this choice to keep the presentation of the algorithms easier and to better understand the benefits of introducing weights in the WILB.

2.2 Weighted iterated local branching heuristic

The WILB takes into consideration that changing the value of some variables or groups of variables is more promising when searching for better solutions than it is for other variables. It also considers the impact of changing the value of a variable on the structure of the solution. In the LBH described in the previous subsection, the local branching constraint limits the total number of variables that can be flipped, and at most Δ variables can flip simultaneously. Using such a constraint, no distinction is made between variables or groups of variables and each variable has the same restriction on being flipped. However, if we can isolate groups of binary variables that are more likely to be flipped when searching for better solutions, this can be exploited to improve the search as an intensification scheme. Isolating groups of variables that by flipping their value may have a large impact in the change of the structure of the current solution, on the other hand, can be exploited as a diversification scheme.

2.2.1 Variable groups and neighborhood definition

We consider three groups of binary variables, defined through weights associated with each variable. When going from the current solution to a better solution in the current neighborhood, each variable is evaluated based on how likely it is that the variable will be flipped to reach the best neighboring solution, and then this evaluation is used to classify variables into three groups. The first group of variables (G_1) is composed of variables that are considered most likely to be flipped when moving to an improving solution within the neighborhood of the current solution. The second group (G_2) is composed of variables that are not strongly suspected to either be more or less likely to change in the current iteration of the search, whereas the third group (G_3) is composed of variables that by changing their value the objective function value of the next solution will likely worsen or the structure of the solution will change significantly. These three

groups form a partition of the set of variables, $N = G_1 \bigcup G_2 \bigcup G_3$, and are defined at each iteration of the WILB. We associate an independent local branching constraint to each group. We expect that the variables in the group G_1 are more promising to change when searching for better solutions than the variables in groups G_2 and G_3. Therefore, we establish that the maximum number of variables that are allowed to flip their value in this group is greater than in the other two groups. Each local branching constraint has a different right hand side parameter according to the expected impact of the corresponding group in the search for better solutions.

The local branching constraints associated with each group are as follows:

$$LB^{\Delta}_{G_1} := \sum_{j \in G_1 \mid \bar{x}_j = 0} x_j + \sum_{j \in G_1 \mid \bar{x}_j = 1} (1 - x_j) \leq \delta_1 \Delta,$$

$$LB^{\Delta}_{G_2} := \sum_{j \in G_2 \mid \bar{x}_j = 0} x_j + \sum_{j \in G_2 \mid \bar{x}_j = 1} (1 - x_j) \leq \delta_2 \Delta,$$

$$LB^{\Delta}_{G_3} := \sum_{j \in G_3 \mid \bar{x}_j = 0} x_j + \sum_{j \in G_3 \mid \bar{x}_j = 1} (1 - x_j) \leq \delta_3 \Delta.$$

where $\delta_1 \geq \delta_2 \geq \delta_3$ are parameters that allow to control the neighborhood of the current solution by providing different flexibility for the change of the values within each group of variables. When $\delta_1 + \delta_2 + \delta_3 > 1$, the three constraints lead to neighborhoods that allow more binary variables to change in any given iteration than allowed by the corresponding local branching constraint (2.2) for a given value of Δ. However, the neighborhoods may still be smaller than the corresponding neighborhoods implied by local branching constraint (2.2), since the combinations of variables that change are more restricted. That is, the size of the neighborhood is being controlled since some combinations of non-promising variables defined by constraint (2.2) are no longer allowed. This highlights the benefit of the introduced variable groups: the solution is allowed to change more in each iteration, even when the neighborhood is not increased in size.

We expect that the variables in group G_3 are the ones leading to the biggest changes in the structure of the obtained solutions when their value flips. Hence, such a group of variables can also be used to explore different regions of the search space and thereby to escape from local optima. To guarantee this, we can force one variable in group G_3 to change by using the following constraint:

$$\overline{LB}_{G_3} := \sum_{j \in G_3 \mid \bar{x}_j = 0} x_i + \sum_{j \in G_3 \mid \bar{x}_j = 1} (1 - x_j) = 1.$$

2.2.2 Overall structure

The general description of the WILB procedure is given in Algorithm 2.

Algorithm 2 Weighted Iterated Local Branching

Let $(\mathbf{x}^0, \mathbf{y}^0)$ be an initial feasible solution and Δ_0, δ_1, δ_2, and δ_3 be given parameters
set $(\bar{\mathbf{x}}, \bar{\mathbf{y}}) := (\mathbf{x}^0, \mathbf{y}^0)$, $f^{Best} := f(\mathbf{x}^0, \mathbf{y}^0)$, $\Delta := \Delta_0$, and $irreg\_iter := false$
repeat
  set $f^{Previous} := f(\bar{\mathbf{x}}, \bar{\mathbf{y}})$
  compute the weight for each variable x_j, $j \in N$
  divide variables $\mathbf{x}$ into groups G_1, G_2, and G_3 according to their weights
  start with the pure MIP model and add the local branching constraints $LB^{\Delta}_{G_1}$, $LB^{\Delta}_{G_2}$ and $LB^{\Delta}_{G_3}$
  if $irreg\_iter$ **then**
    add constraint $\overline{LB}_{G_3}$ to the model (optional)
  end if
  run the MIP solver to obtain a new solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$
  if $f(\bar{\mathbf{x}}, \bar{\mathbf{y}}) > f^{Best}$ **then**
    set $f^{Best} := f(\bar{\mathbf{x}}, \bar{\mathbf{y}})$
  end if
  if $f(\bar{\mathbf{x}}, \bar{\mathbf{y}}) > f^{Previous}$ **then**
    $irreg\_iter := false$
    $\Delta := \Delta_0$
    go to Step 3
  else
    $irreg\_iter := true$
    increase Δ and go to Step 3
  end if
until a given stopping criterion is met

The structure of the WILB presented in Algorithm 2 is similar to the structure of the LBH in Algorithm 1. The WILB has two additional steps (Steps 5 and 6) to form the partition of variables. In Step 5 we compute the weight for each binary variable x_j, $i \in N$. The computation depends on the problem being solved. In Step 6, the variables are divided into three groups according to their calculated weights. We associate variables with lower weights with group G_1, while variables with higher weights are associated with group G_3. All the remaining variables are included in group G_2. After grouping all the variables, a local branching constraint for each group is added to the model (Step 7). There is also an irregular iteration, performed at the optional step (Step 9) of adding the local branching constraint $\overline{LB}_{G_3}$ to the model. This helps to direct the search towards a new region of the search space.

2.2.3 Determination of weights and constitution of groups

We now introduce different ways of computing weights for the binary variables $\mathbf{x}$ and explain how to form the three groups of such variables. Consider an incumbent feasible solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ for the general MIP problem (2.1). We first define weights for each binary variable $\mathbf{x}$, such that lower weights are associated with variables that we believe are more likely to flip when moving from the current solution to an improving solution in the current neighborhood. To derive weights, we consider three different factors described as follows.

The first factor (F_1) is based on the objective function coefficient of each variable. Since it is more beneficial to set variables to one if their objective function coefficients

are higher, the weight of variable x_j, $j \in N$, for this factor is defined as follows:

$$w_j^{F_1} = \bar{x}_j + (-1)^{\bar{x}_j} \left(\frac{C^{MAX} - c_j}{C^{MAX} - C^{MIN}} \right),$$

where C^{MAX} and C^{MIN} are the values of the highest and lowest objective function coefficients among all variables and $\bar{x}_j$ is the current value of variable x_j in the incumbent solution. Weights $w_j^{F_1}$ are therefore in the range [0, 1] for all $j \in N$. In an incumbent solution, the binary variables we believe are more likely to be flipped are either the ones having a current value of one and a small objective function coefficient or the ones having a current value of zero and a high objective function coefficient. Both cases lead to a small weight $w_j^{F_1}$ for the associated variable.

The second factor (F_2) is related to the signals of the coefficients of each variable in the functional constraints in which it appears. In problem (2.1), any functional constraint $g_i(\mathbf{x}, \mathbf{y}) \leq 0$ for $i \in I$ is linear, and therefore it has the form $\sum_{j=1}^{n} a_{ij} x_j + \sum_{j=1}^{p+q} b_{ij} y_j \leq 0$, with $a_{ij}, b_{ij} \in \mathbb{R}$. By taking $x_j = 1$ if $a_{ij} < 0$ and $x_j = 0$ if $a_{ij} > 0$, the left-hand side value of constraint i decreases and therefore such constraint becomes easier to satisfy for the value of the remaining variables. This can be seen as a *smooth* relaxation of constraint i with respect to the variable x_j. Let us denote by S_j^+ (respectively S_j^-) the number of constraints in which variable x_j appears with a positive (respectively negative) coefficient. A variable with a high difference $D_j := S_j^+ - S_j^-$ more often appears with positive coefficients in the functional constraints than with negative coefficients. This means that the number of constraints *smoothly* relaxed when such a variable is fixed to zero is larger than the number of constraints *smoothly* relaxed when it is fixed to one. Hence, for such a variable, taking the value zero is beneficial when searching for new feasible solutions, as this ensures that a large number of constraints are *smoothly* relaxed. The weight of variable x_j, $j \in N$, for this factor is therefore defined as follows:

$$w_j^{F_2} = (1 - \bar{x}_j) + (-1)^{(1-\bar{x}_j)} \left(\frac{D^{MAX} - D_j}{D^{MAX} - D^{MIN}} \right),$$

where $D^{MAX} := \max_{j \in N} \{D_j\}$ and $D^{MIN} := \min_{j \in N} \{D_j\}$. The weight $w_j^{F_2}$ takes values in the interval [0, 1] for all $j \in N$. Again, small weights are associated with the variables that are considered promising to flip in order to discover an improved solution. Such variables are the ones having either $\bar{x}_j = 0$ and a small value of D_j or $\bar{x}_j = 1$ and a high value of D_j.

The third factor (F_3) is the number of constraints that would be violated if a single variable is flipped in the current solution while keeping the current value of the remaining variables fixed. The variables that have the lowest number of constraints that become violated are more promising to change in the pursuit of an improved neighboring solution, while those with large number of constraints becoming violated will force a larger change in the structure of the next solution in case their value is

flipped. Hence, the weight of variable x_j, $j \in N$, for this factor is defined as follows:

$$w_j^{F_3} = 1 - \left(\frac{V^{MAX} - V_j}{V^{MAX} - V^{MIN}} \right)$$

where $V^{MAX} := \max_{j \in N}\{V_j\}$, $V^{MIN} := \min_{j \in N}\{V_j\}$ and V_j is the number of constraints that are violated when variable x_j changes its value in the incumbent solution and the remaining variables keep their value fixed. The weights $w_j^{F_3}$ also vary between 0 and 1.

The three factors previously defined are general for any MIP problem with binary variables. For defining a unique weight for each variable, the three factors can either be used separately or in combination. Since the weights associated to the three factors are normalized, vary in the same interval [0, 1], and smaller values are related with variables that are promising to change without changing the structure of the solution significantly, a combination of these factor-weights can easily be employed by simply adding the factor-weights. Combining these three factor-weights when considering a general MIP problem has several advantages. First, the three factor-weights are based on different features of the problem, which allows to determine a more holistic ranking of the variables. Second, the combination of factor-weights helps to better distinguish between promising and non-promising variables, especially when some of the factor-weights do not lead to large weight variations when applied separately. While the three factor-weights presented are general for any MIP problem, different factor-weights can be derived for specific MIP problems by taking into account their structural characteristics.

After calculating a weight for each binary variable, three groups of variables are created: G_1, consisting of the most promising variables to change; G_2 with the variables that do not have a strong inclination towards or against being flipped; and G_3, containing those variables whose change will probably lead to an increase of cost or promote a large change in the structure of the solution. By construction, the variables are more promising to change the lower their weight. Hence, we include in group G_1 the variables with smallest weights, include in group G_3 the variables with highest weights, and include in group G_2 the remaining variables.

The number of variables to include in each group can be determined by a fixed percentage, defined *a priori*, of the total number of binary variables. Therefore, we consider that each of the three groups has a fixed percentage of variables, which is $\alpha_1\%$, $(100 - \alpha_1 - \alpha_2)\%$, and $\alpha_2\%$, respectively. To achieve good performances in terms of the computational time, the percentage of variables in group G_1 should be small, since the constraint LB_{G1}^{Δ} has the highest right-hand side value given our choice of $\delta_1 \geq \delta_2 \geq \delta_3$. Furthermore, since group G_3 is the one where a lower number of variables is allowed to change, the number of variables in this group should also be small to reduce the chances of being trapped in a local optimum.

3 The optimum satisfiability problem

We test the performance of the proposed WILB procedure on the OptSAT. Informally, this problem can be characterized as a satisfiability problem with an added objective function consisting of a weighted sum of the truth values of the variables. When this objective function is maximized, the problem was originally called the Boolean optimization problem (Davoine et al. 2003), while the variant where the objective function is minimized is also known as the minimum-cost satisfiability problem (Li 2004).

The OptSAT subsumes several well-known and NP-hard binary optimization problems, such as weighted versions of set covering, graph stability, set partitioning, and maximum satisfiability problems. For some of these problems, a conversion to OptSAT requires the introduction of additional variables (Li 2004). As an example, in (partial) maximum satisfiability (Li and Manyà 2009), a slack variable must be added for each soft clause (Hvattum et al. 2006). The OptSAT can be formulated as follows:

$$\begin{aligned} \max \quad & \sum_{i \in N} c_j x_j \\ s.t. \quad & g(x_1, \ldots, x_n) = True, \\ & x_j \in \{0, 1\}, \qquad\qquad j \in N, \end{aligned}$$

where $g(x_1, \ldots, x_n) = C_1 \wedge \ldots \wedge C_m = True$ is a Boolean equation written in conjunctive normal form. Each clause C_i, $i = 1, ..., m$ is a disjunction of some non-negated variables and some negated variables. Denoting by A_i and B_i, respectively, the sets of indices of negated and non-negated variables in the clause C_i, such a clause can be written as $C_i = \left(\bigvee_{j \in A_i} \neg x_j\right) \vee \left(\bigvee_{j \in B_i} x_j\right)$. The OptSAT problem can therefore be written as a mixed integer programming problem as follows (Hvattum et al. 2004):

$$\begin{aligned} \max \quad & \sum_{j \in N} c_j x_j \\ s.t. \quad & \sum_{j \in B_i} x_j + \sum_{j \in A_i} (1 - x_j) \geq 1, \qquad i = 1, ..., m \\ & x_j \in \{0, 1\}, \quad j \in N \end{aligned} \tag{3.1}$$

This is the formulation of the OptSAT problem that we use in our computational experiments.

4 Computational results

This section describes the computational experiments carried out to compare the performance of the proposed WILB against a non-weighted LBH. The LBH corresponds to the Algorithm 1, while the WILB corresponds to Algorithm 2. All tests were run using a computer with an Intel Core i7-4750HQ 2.00 GHz processor and 8 GB of

RAM, and were conducted using the Xpress-Optimizer 8.6.0 solver with the default options.

We perform our main experiments on a subset of OptSAT instances used in Davoine et al. (2003). The full set is composed of 63 classes of instances, however, we only consider a subset K of 13 of those classes: $K = \{27, 29, 31, 38, 40, 49, 52, 53, 54, 59, 61, 62, 63\}$. The classes in K are those that are the most difficult to solve for MIP solvers (Hvattum et al. 2012). The first six classes (27, 29, 31, 38, 40, and 49) consist of randomly generated instances, classes 52, 53, and 54 are formed by instances from the graph stability problem, and the remaining classes (59, 61, 62, and 63) correspond to instances from the set covering problem. Thus, while the WILB is only tested on the OptSAT, the structure of the instances solved are also representative for graph stability and set covering instances.

From each class we consider five instances. A training set is used to tune the value of several parameters. It is composed of 13 instances, corresponding to the first instance of each class (with label *num0*). A test set is used for comparison of the proposed WILB against the classic LBH, consisting of the remaining 52 instances (last four instances of each class (*num1* to *num4*). To obtain the initial solutions used in both LBH and WILB, we run the MIP model with no local branching constraints for 20 seconds.

In an additional experiment, we use larger test instances presented in da Silva et al. (2020). There are six classes of instances, $K' = \{64, 65, \ldots, 69\}$, with the number of binary variables varying between 500 and 3000, and the number of constraints (clauses) varying between 2500 and 15000. These instances form a secondary test set, with instances that are different in structure compared to the instances in the training set.

4.1 Calibration tests

In this section, we present all the preliminary tests performed on the training set of 13 instances to tune the value of several parameters. We start by calibrating the parameters of the classic LBH (that are common to the WILB) and then the specific parameters of the WILB.

4.1.1 Calibration tests for the LBH

The LBH has four parameters: the value Δ_0 to define the local branching constraint, the increment on the Δ used when irregular iterations are performed (Step 14 of Algorithm 1), the stopping criterion used when solving each subproblem, and the global stopping criterion.

We set the increment to $\Delta_0/2$, and for the parameter Δ_0 we test the values 5, 10, 15, and 20. As remarked by Fischetti and Lodi (2003), values of Δ_0 in range [10, 20] proved effective in most cases. Furthermore, the value of Δ_0 increases when no improved solutions are found, making it less useful to test even larger values of Δ_0 than 20.

Table 1 Results for all combinations of the LBH parameters

	TL=30		TL=60		TL=120	
Δ_0	Avg.	NBS	Avg.	NBS	Avg.	NBS
5	108,715	2	108,754	4	108,770	4
10	108,685	2	108,777	9	108,755	7
15	108,690	2	108,724	3	108,727	5
20	108,718	5	108,724	7	108,759	6

The termination criterion used when solving each subproblem and the global stopping criterion are both defined through a time limit. The global stopping criterion is 600 seconds (10 minutes), and we tested the values 30, 60, and 120 seconds (corresponding to short, medium and large computational times) for the time limit (TL) used for solving each subproblem. The solution of a subproblem is also halted in the case that a proven optimal solution for the given neighborhood is obtained. Table 1 shows the results for each combination of Δ_0 and TL. The columns *Avg.* report the average objective function value of the obtained solutions, while the columns *NBS* display the number of best solutions found by each combination of parameters, out of a total of 13 instances.

Table 1 shows that the best (highest) average objective function value is associated with the solutions obtained using the combination $\Delta_0 = 10$ and TL=60s. This combination also leads to the highest number of best feasible solutions found. These results indicate the superiority of this combination of parameters. Thus, in what follows we use $\Delta_0 = 10$ and TL=60s.

4.1.2 Calibration tests for the WILB

Once the parameters for LBH had been determined, the parameters for WILB were tuned. All parameters that are in common for both methods were kept at the values found when tuning LBH ($\Delta_0 = 10$, increment = $\Delta_0/2$, TL = 60s, and global time limit = 600s), and only the parameters specific to WILB were considered next. This includes: the values of δ_1, δ_2, and δ_3, the percentages α_1 and α_2, and the best way of combining weight factors.

Taking into account the definition and interpretation of the groups G_1, G_2 and G_3, as well as the need of relating both WILB and LBH, setting $\delta_2 = 1$ in the local branching constraint $LB^{\Delta}_{G_2}$ is a natural choice. By construction, $LB^{\Delta}_{G_2}$ is the local branching constraint involving the largest number of binary variables. The local branching constraints $LB^{\Delta}_{G_1}$ and $LB^{\Delta}_{G_3}$ are associated with the variables that are, respectively, more promising and less promising to flip within the neighborhood. Hence, $\delta_1 > \delta_2 > \delta_3$, and therefore, we use $\delta_1 = 2$, $\delta_2 = 1$ and $\delta_3 = 1/2$ for constraints $LB^{\Delta}_{G_1}$, $LB^{\Delta}_{G_2}$, and $LB^{\Delta}_{G_3}$, respectively.

In the WILB we also have to choose the best way of combining weight factors and the percentages of variables (defined by parameters α_1 and α_2) to include in each one of the three groups: G_1, G_2, and G_3. The factor-weights were defined in Section 2.2.3. Regarding the second factor-weight, for the OptSAT problem the number of

Table 2 Results for all combinations of the WILB parameters

	$w_j^{F_1}$		$w_j^{F_2}$		$w_j^{F_3}$	
(α_1, α_2)	Avg.	NBS	Avg.	NBS	Avg.	NBS
(10,10)	108,710	2	108,766	3	108,772	4
(10,20)	108,718	0	108,741	2	108,747	2
(20,10)	108,674	2	108,754	3	108,786	4
(20,20)	108,653	0	108,744	2	108,744	3
	$w_j^{F_1} + w_j^{F_2}$		$w_j^{F_1} + w_j^{F_3}$		$w_j^{F_2} + w_j^{F_3}$	
(α_1, α_2)	Avg.	NBS	Avg.	NBS	Avg.	NBS
(10,10)	108,798	4	108,811	4	108,754	3
(10,20)	108,778	4	108,781	4	108,754	3
(20,10)	108,795	4	108,911	4	108,754	3
(20,20)	108,787	4	108,786	4	108,754	3
	$w_j^{F_1} + w_j^{F_2} + w_j^{F_3}$					
(α_1, α_2)	Avg.	NBS				
(10,10)	108,818	3				
(10,20)	108,886	3				
(20,10)	108,947	5				
(20,20)	108,937	4				

constraints in which a given binary variable x_j appears with a positive (resp. negative) coefficient is exactly the cardinality of the set A_j (resp. B_j), that is, $S_j^+ := |A_j|$ and $S_j^- := |B_j|$. We consider seven different ways of defining weights based on the different combinations of the three factors: F_1, F_2, and F_3. For the parameters (α_1,α_2), we consider four different combinations of values. In these calibration experiments, we do not use the optional step, Step 9, of Algorithm 2. The obtained results are summarized in Table 2.

The results in Table 2 show that the variations between the combinations tested are not large. However, since the highest number of best solutions was found by using the weights $w_j = w_j^{F_1} + w_j^{F_2} + w_j^{F_3}$ and $(\alpha_1, \alpha_2) = (20\%, 10\%)$ this becomes the combination used to test the WILB. This combination also leads to the highest average objective function value of the solutions.

4.2 Main results

We now present the main results to assess the performance of the proposed WILB. First, we introduce some performance metrics and then we compare the WILB (defined by Algorithm 2) against the LBH (defined by Algorithm 1). After that, we compare the WILB against solving the problem directly using the Xpress solver with an imposed

time limit, and finally we compare the WILB against reference values of the literature obtained with CPLEX.

For comparing performances, we use two metrics that are independently computed for each class of instances. For each instance $h \in \{1, 2, 3, 4\}$ in class $k \in K$, denote by z_{hk}^{LBH} and z_{hk}^{WILB} the objective function values of the solutions obtained by LBH and WILB, respectively. The first metric (da Silva et al. 2020) is defined as

$$M_k^1 = \frac{100}{4} \sum_{h=1}^{4} \frac{z_{hk}^{WILB}}{z_{hk}^{LBH}}, \quad k \in K.$$

Values of this metric greater than 100, indicate a better average performance of the WILB comparing to the LBH. The second metric (Rodrigues et al. 2021) provides a global perspective on the number of best solutions found by the WILB comparing to the LBH. To calculate this value, we start by computing for each instance $h \in \{1, 2, 3, 4\}$ in class $k \in K$ the value

$$s_{hk} = \begin{cases} -1, & \text{if } z_{hk}^{WILB} < z_{hk}^{LBH}, \\ 0, & \text{if } z_{hk}^{WILB} = z_{hk}^{LBH}, \\ 1, & \text{if } z_{hk}^{WILB} > z_{hk}^{LBH}. \end{cases}$$

Then, the value of the metric for each class is defined as

$$M_k^2 = \frac{1}{4} \sum_{h=1}^{4} s_{hk}.$$

The value M_k^2 of this metric varies between -4 and 4. When $M_k^2 = 4$, this metric shows an absolutely better performance of the WILB, meaning that the solutions obtained for the four instances of class k are all better than the ones obtained by the LBH.

Now we present the values of both metrics for all the classes of instances tested. Recall that the LBH has the following settings: $\Delta_0 = 10$; the irregular iterations are performed by redefining $\Delta := \Delta + \Delta_0/2$; the global total time limit is 600 seconds; and the time limit for solving each MIP model is 60 seconds. The WILB uses the same value of Δ as the LBH, and the maximum number of variables allowed to flip their value in each group G_1, G_2, and G_3 is 2Δ, Δ, and $\Delta/2$, respectively. Moreover, the weights are determined according to the formula $w_j = w_j^{F_1} + w_j^{F_2} + w_j^{F_3}$ and the percentages of variable in groups G_1, G_2, and G_3 are 20%, 70%, and 10%, respectively. The comparison between this WILB version against the standard LBH using the metrics described before is displayed in Table 3. We also consider a version of WILB that performs the optional Step 9 of Algorithm 2, which we refer to as WILB$^{\overline{LB}}$. This optional step is not performed in the simple WILB version.

Furthermore, Table 3 also shows results for two additional versions of the WILB: WILB$_L$ and WILB$_L^{\overline{LB}}$. In these versions, smaller right hand side coefficients are defined

Table 3 Comparison between the WILB versions and the classic LBH

	WILB		WILB$^{\overline{LB}}$		WILB$_L$		WILB$_L^{\overline{LB}}$	
Class k	M_k^1	M_k^2	M_k^1	M_k^2	M_k^1	M_k^2	M_k^1	M_k^2
27	99.98	−1	100.00	1	99.99	−1	100.01	1
29	100.01	0	100.03	1	100.03	1	100.05	2
31	100.11	2	100.13	4	100.09	4	100.07	4
38	99.99	1	100.01	2	100.00	2	100.01	0
40	100.06	1	100.07	3	100.06	3	100.06	3
49	100.07	4	100.02	2	100.10	4	100.09	3
52	100.02	1	100.26	2	100.02	1	100.38	2
53	100.53	2	100.53	2	100.39	2	100.76	2
54	102.15	4	102.15	4	101.58	2	101.58	2
59	99.98	−1	100.00	0	100.00	0	100.00	0
61	100.03	4	100.00	2	100.00	0	100.03	0
62	100.02	1	100.00	1	100.01	2	100.02	3
63	100.10	4	100.10	4	100.11	4	100.11	4
Average	100.23	1.7	100.25	2.2	100.18	1.8	100.24	2.0

for constraints $LB^{\Delta}_{G_1}$, $LB^{\Delta}_{G_2}$, and $LB^{\Delta}_{G_3}$, using respectively $\delta_1 = 1$, $\delta_2 = 0.5$, and $\delta_3 = 0.25$.

From Table 3, we observe the superiority of the WILB against the LBH since for most of the classes the values M_k^1 are greater than 100. Generally, the values of M_k^2 are positive, which reinforces this conclusion. Another conclusion that can be drawn from Table 3 is that the use of the local branching constraint $\overline{LB}$ in the irregular iterations, corresponding to the optional Step 9 of Algorithm 2 used in versions WILB$^{\overline{LB}}$ and WILB$_L^{\overline{LB}}$, substantially improves the performance of the WILB. The best global average values are associated with the version WILB$^{\overline{LB}}$. There are some classes of instances where $M_k^2 = 4$, indicating that the WILB version used was able to obtain better solutions than the ones obtained by the LBH for all the instances of that class. Finally, both WILB and WILB$_L$ obtained a worse solution than the one obtained by LBH in only 7 of the 52 instances tested, while the WILB$^{\overline{LB}}$ and WILB$_L^{\overline{LB}}$ versions obtained worse solutions for only 4 of the instances.

The average improvement between the four WILB variants and the LBH is around 0.2% and this improvement is significant since the optimality gaps of the obtained solutions are, in general, very low. Except for the classes 52, 53, and 54 (where the average optimality gap is about 12%, 25%, and 36%) in all the remaining instances, the optimality gaps are lower than 3% and the average optimality gap in all those instances is about 1.7%. To compute the optimality gaps, we solved the simple model with Xpress for 600 seconds to obtain an upper bound for each instance.

To better understand the behaviour of the WILB, Table 4 reports the rounded average number of subproblems (NS) solved with the imposed time limit, the rounded

percentage of such problems solved to optimality ($\%opt$), and the average optimality gap associated with all subproblems solved (in percentage).

The obtained results show that three of the four WILB versions tested allow solving a larger number of subproblems (as well as to obtain a larger percentage of such subproblems solved to optimality) than the classic LBH. We can also see that the average gaps obtained by WILB_L and $\text{WILB}_L^{\overline{LB}}$ (where smaller neighbourhoods are defined) are lower than the ones obtained by the classic LBH. Hence, the results of both Tables 3 and 4 allow to conclude that the WILB is efficient when defining neighbourhoods of incumbent solutions.

We also perform experiments to compare the solutions obtained by WILB against the solutions obtained by solving the OptSAT problem directly using the solver Xpress with a time limit of 600+20=620 seconds. We use the same metrics defined before, replacing z_{hk}^{LBH} by z_{hk}^{Xpress}, where z_{hk}^{Xpress} is the objective function value obtained by Xpress for instance h in class k. The obtained results are displayed in Table 5.

The obtained results reveal the superiority of the WILB against Xpress since all the values of M_k^1 are greater than 100 and the values of M_k^2 are all positive and are almost always equal to 4 for all the classes of instances.

Finally, for the sake of backward compatibility, we compare the WILB against reference values obtained by CPLEX as reported in Davoine et al. (2003). We use the same metrics defined before, replacing z_{hk}^{LBH} by z_{hk}^{CPLEX}, where z_{hk}^{CPLEX} is the reference value reported in Davoine et al. (2003) for instance h in class k. The obtained results are displayed in Table 6.

As shown in Table 6, all the solutions obtained by WILB are better than the ones obtained by CPLEX and the average improvement is about 4.81%.

4.3 Results for large size instances

In this section, we test the performance of the WILB against the LBH by using a set of large size instances of the OptSAT problem: classes labeled from 64 to 69. We did not perform any calibration tests on these instances, and therefore, the configuration of both WILB and LBH is exactly the same as before. We also use the same two metrics and the obtained results are reported in Table 7. The obtained results show that the proposed WILB outperforms the classic LBH also when large size instances are considered.

5 Conclusion

This paper introduces a new local-search based matheuristic referred to as weighted iterated local branching (WILB). The WILB is designed under the assumption that to improve a given feasible solution not all binary variables are equally likely to be flipped in a given iteration. The method consists of defining different groups of binary variables and creating an independent local branching constraint for each of them. Each local branching constraint has a different right-hand side value, defined according to the expected contribution of the associated variables for improving the incumbent

Table 4 Additional statistics for accessing the behaviour of the WILB

	LBH			WILB			$\text{WILB}^{\overline{LB}}$			WILB_L			$\text{WILB}_L^{\overline{LB}}$		
Class k	NS	$\%opt$	Gap	NS	$\%opt$	Gap	NS	$\%opt$	Gap	NS	$\%opt$	Gap	NS	$\%opt$	Gap
27	19	52	1.2	14	27	2.5	24	72	0.9	18	48	1.4	29	75	0.6
29	18	49	0.4	18	48	0.4	27	68	0.3	27	70	0.1	30	73	0.1
31	18	39	0.6	12	33	0.7	21	61	0.4	17	54	0.3	21	61	0.2
38	18	49	0.3	16	40	0.4	28	68	0.2	21	60	0.2	26	68	0.1
40	17	56	0.4	14	38	0.5	27	67	0.4	22	64	0.2	25	69	0.2
49	18	47	0.6	15	37	0.6	28	70	0.4	21	62	0.2	23	66	0.2
52	18	70	1.0	20	59	2.0	20	60	1.4	29	78	0.6	30	82	0.4
53	17	69	2.2	17	48	5.2	16	43	4.3	23	67	2.0	23	68	1.7
54	15	52	0.9	11	22	1.7	11	22	1.7	20	88	0.3	20	88	0.3
59	14	42	1.9	13	24	3.2	15	47	1.2	19	47	1.8	22	66	0.7
61	15	56	0.4	20	62	0.3	24	69	0.3	28	73	0.1	31	76	0.1
62	15	30	0.3	14	30	0.3	19	49	0.2	18	51	0.1	23	59	0.1
63	16	47	0.6	11	11	0.6	17	40	0.4	17	59	0.3	21	62	0.3
Average	17	51	0.8	15	37	1.4	21	57	0.9	21	63	0.6	25	70	0.4

Table 5 Comparison between the WILB variants and the Xpress solver

	WILB		WILB$^{\overline{LB}}$		WILB$_L$		WILB$_L^{\overline{LB}}$	
Class k	M_k^1	M_k^2	M_k^1	M_k^2	M_k^1	M_k^2	M_k^1	M_k^2
27	100.25	4	100.27	4	100.27	4	100.29	4
29	100.29	3	100.31	3	100.31	4	100.33	4
31	100.84	4	100.86	4	100.82	4	100.80	4
38	100.15	4	100.18	4	100.16	4	100.17	4
40	101.02	4	101.02	4	101.02	4	101.02	4
49	100.76	4	100.72	4	100.79	4	100.79	4
52	102.33	4	102.59	4	102.33	4	102.71	4
53	107.94	4	107.94	4	107.80	4	108.19	4
54	111.65	4	111.65	4	111.04	4	111.04	4
59	100.52	4	100.55	4	100.55	4	100.55	4
61	100.66	4	100.63	4	100.62	4	100.66	4
62	100.02	2	100.01	0	100.01	2	100.02	4
63	101.25	4	101.25	4	101.25	4	101.26	4
Average	102.13	3.8	102.15	3.6	102.07	3.8	102.14	4.0

Table 6 Comparison between the WILB variants and the CPLEX solver

	WILB		WILB$^{\overline{LB}}$		WILB$_L$		WILB$_L^{\overline{LB}}$	
Class k	M_k^1	M_k^2	M_k^1	M_k^2	M_k^1	M_k^2	M_k^1	M_k^2
27	110.01	4	110.03	4	110.03	4	110.05	4
29	101.40	4	101.40	4	101.40	4	101.42	4
31	101.34	4	101.36	4	101.32	4	101.30	4
38	100.88	4	100.91	4	100.89	4	100.90	4
40	101.59	4	101.60	4	101.59	4	101.59	4
49	101.47	4	101.43	4	101.50	4	101.50	4
52	109.67	4	109.93	4	109.67	4	110.07	4
53	111.87	4	111.87	4	111.72	4	112.12	4
54	112.43	4	112.43	4	111.81	4	111.81	4
59	107.98	4	108.01	4	108.01	4	108.01	4
61	101.48	4	101.46	4	101.45	4	101.49	4
62	101.11	4	101.09	4	101.09	4	101.11	4
63	101.30	4	101.30	4	101.30	4	101.31	4
Average	104.81	4.0	104.83	4.0	104.75	4.0	104.82	4.0

Table 7 Comparison between the WILB variants and LBH for large size instances

	WILB		WILB$^{\overline{LB}}$		WILB$_L$		WILB$_L^{\overline{LB}}$	
Class k	M_k^1	M_k^2	M_k^1	M_k^2	M_k^1	M_k^2	M_k^1	M_k^2
64	100.45	2	100.49	2	100.28	2	100.54	2
65	100.95	2	101.32	4	101.07	4	101.07	4
66	101.34	4	101.34	4	100.46	4	100.46	4
67	101.59	4	101.58	4	100.43	4	100.41	4
68	101.22	4	101.26	4	100.22	2	100.27	2
69	101.55	4	101.58	4	100.70	4	100.71	4
Average	101.18	3.3	101.26	3.7	100.53	3.3	100.58	3.3

solution. The construction of the groups considers weights computed for each variable using three factors. In the WILB, the set of variables that are less promising to flip can also be used to direct the local search into different regions of the search space every time the obtained solution is not better than the previous one.

Computational experiments carried out on benchmark instances of the optimum satisfiability problem (OptSAT) show a clear advantage of using this WILB procedure compared to an unweighted local branching heuristic (LBH). The experiments also show the benefits of using the non-promising variables to perform irregular iterations and escape from local optima. In the main test, the WILB obtained better results than the standard LBH for around 61% of the instances and worse results for around 9% of the instances. The solutions obtained by both procedures are identical for approximately 30% of the instances. It also becomes clear from the computational results that the proposed WILB completely outperforms the approach of solving the problem directly by using Xpress with a time limit. Indeed, the WILB gets around 97% better solutions than Xpress, 1% of equal solutions, and 2% of worse solutions. Furthermore, in an additional experiment using larger test instances, WILB is better than LBH on almost all of the instances.

The new WILB was tested only on a single problem class, the OptSAT. However, the variety of instances used is large, ranging from set covering instances and graph stability instances to randomly generated instances. Thus, given the dominance of the WILB compared to an unweighted version, it seems clear that the WILB is a promising approach to solving hard optimization problems having many binary variables.

The third weight factor in the WILB considers the number of constraints that become violated if flipping a given variable. In OptSAT all the left-hand-side coefficients are -1, 0, or 1. However, for more general problems, the coefficients may have a wider range of values. In that case, it may be better for the third weight factor to consider the amount by which constraints are violated, rather than just the number of violated constraints. Future research is needed to examine this issue, which may require that constraints are normalized before evaluating the importance of given variables.

Acknowledgements The research was partially supported by the Center for Research and Development in Mathematics and Applications (CIDMA) through the Portuguese Foundation for Science and Technology

(FCT - Fundação para a Ciência e a Tecnologia), references UIDB/04106/2020 and UIDP/04106/2020. The research of the first author was also partially supported by the Project CEMAPRE/REM - UIDB/05069/2020 - financed by FCT/MCTES through national funds. The third author was supported by the AXIOM project, partially funded by the Research Council of Norway.

Funding Open access funding provided by Molde University College - Specialized University in Logistics.

References

Alsheddy, A., Voudouris, C., Tsang, E.P.K., Alhindi, A.: Guided local search. In: Martí, R., Panos, P., Resende, M.G. (eds.) Handbook of Heuristics, pp. 1–37. Springer International Publishing, Cham (2016)

Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys **35**(3), 268–308 (2003)

Boussaïd, I., Lepagnot, J., Siarry, P.: A survey on optimization metaheuristics. Information Sciences **237**, 82–117 (2013)

da Silva, R., Hvattum, L.M., Glover, F.: Combining solutions of the optimum satisfiability problem using evolutionary tunneling. MENDEL Soft Computing Journal **26**(1), 23–29 (2020)

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

Dauer, A.T., de Athayde Prata, B.: Variable fixing heuristics for solving multiple depot vehicle scheduling problem with heterogeneous fleet and time windows. *Optimization Letters*, 15(1):153–170, (2021)

Davoine, T., Hammer, P., Vizvári, B.: A heuristic for Boolean optimization problems. Journal of Heuristics **9**, 229–247 (2003)

Faria, A., de Souza, S., de Sá, E., Silva, C.: Variable neighborhood descent branching applied to the multi-way number partitioning problem. Electronic Notes in Theoretical Computer Science **346**, 437–447 (2019)

Fischetti, M., Lodi, A.: Local branching. Mathematical Programming **98**(1–3), 23–47 (2003)

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

Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publisher, Boston, Dordrecht, London (1997)

González, P.H., Simonetti, L., Michelon, P., Martinhon, C., Santos, E.: A variable fixing heuristic with local branching for the fixed charge uncapacitated network design problem with user-optimal flow. Computers & Operations Research **76**, 134–146 (2016)

Hernandez, F., Gendreau, M., Jabali, O., Rei, W.: A local branching matheuristic for the multi-vehicle routing problem with stochastic demands. Journal of Heuristics **25**, 215–245 (2019)

Hill, A., Voß, S.: Generalized local branching heuristics and the capacitated ring tree problem. Discrete Applied Mathematics **242**, 34–52 (2018)

Hvattum, L.M., Løkketangen, A., Glover, F.: New heuristics and adaptive memory procedures for Boolean optimization problems. In: Karlof, J. (ed.) Integer Programming: Theory and Practice, pp. 1–18. CRC Press, Boca Raton, FL (2006)

Hvattum, L.M., Løkketangen, A., Glover, F.: Comparisons of commercial MIP solvers and an adaptive memory (tabu search) procedure for a class of 0–1 integer programming problems. Algorithmic Operations Research **7**, 13–21 (2012)

Hvattum, L.M., Løkketangen, A., Glover, F.: Adaptive memory search for Boolean optimization problems. Discrete Applied Mathematics **142**(1), 99–109 (2004)

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)

Li, C., Manyà, F.: MaxSAT, hard and soft constraints. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 613–631. IOS Press, (2009)

Li, X.: *Optimization Algorithms for the Minimum-Cost Satisfiability Problem*. Phd thesis, North Carolina State University, Raleigh, North Carolina, (2004)

Raidl, G., Puchinger, J.: Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In C. B. C., M. Aguilera, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 114 of *Studies in Computational Intelligence*, pages 31–62. Springer, Berlin, Heidelberg, (2008)

Rei, W., Gendreau, M., Soriana, P.: A hybrid monte carlo local branching algorithm for the single vehicle routing problem with stochastic demands. Transportation Science **44**, 136–146 (2010)

Rodrigues, F., Agra, A., Hvattum, L.M., Requejo, C.: Weighted proximity search. Journal of Heuristics **27**(3), 459–496 (2021)

Rodríguez-Martín, I., Salazar-González, J.: A local branching heuristic for the capacitated fixed-charge network design problem. Computers and Operations Research **37**, 575–581 (2010)

Rothberg, E.: An evolutionary algorithm for polishing mixed integer programming solutions. INFORMS Journal on Computing **19**, 534–541 (2007)

Sadykov, R., Vanderbeck, F., Pessoa, A., Tahiri, I., Uchoa, E.: Primal heuristics for branch and price: The assets of diving methods. INFORMS Journal on Computing **31**(2), 251–267 (2019)

Samavati, M., Essam, D., Nehring, M., Sarker, R.: A local branching heuristic for the open pit mine production scheduling problem. European Journal of Operational Research **257**, 261–271 (2017)

Sarayloo, F., Crainic, T.G., Rei, W.: A reduced cost-based restriction and refinement matheuristic for stochastic network design problem. Journal of Heuristics **27**(3), 325–351 (2021)

Sörensen, K., Glover, F.: Metaheuristics. Scholarpedia **10**(4), 6532 (2015)

Stützle, T., Ruiz, R.: Iterated local search. In: Martí, R., Panos, P., Resende, M.G.C. (eds.) Handbook of Heuristics, pp. 1–27. Springer International Publishing, Cham (2017)

Wang, Y., Lü, Z., Glover, F., Hao, J.-K.: Effective variable fixing and scoring strategies for binary quadratic programming. In P. Merz and J.-K. Hao, editors, *Evolutionary Computation in Combinatorial Optimization*, pages 72–83, Berlin, Heidelberg, (2011). Springer Berlin Heidelberg