The Weighted Independent Domination Problem: ILP Model and Algorithmic Approaches

Pedro Pinacho Davidson[1,2], Christian Blum[3], and José A. Lozano[1,4]

[1]Department of Computer Science and Artifical Intelligence
University of the Basque Country UPV/EHU, San Sebastian, Spain
ja.lozano@ehu.eus

[2]Escuela de Informática
Universidad Santo Tomás, Concepción, Chile
ppinacho@santotomas.cl

[3]Artificial Intelligence Research Institute (IIIA-CSIC)
Campus of the UAB, Bellaterra, Spain
christian.blum@iiia.csic.es

[4]Basque Center for Applied Mathematics (BCAM), Bilbao, Spain

Abstract

This work deals with the so-called weighted independent domination problem, which is an *NP*-hard combinatorial optimization problem in graphs. In contrast to previous work, this paper considers the problem from a non-theoretical perspective. The first contribution consists in the development of three integer linear programming models. Second, two greedy heuristics are proposed. Finally, the last contribution is a population-based iterated greedy metaheuristic which is applied in two different ways: (1) the metaheuristic is applied directly to each problem instance, and (2) the metaheuristic is applied at each iteration of a higher-level framework—known as construct, merge, solve & adapt—to sub-instances of the tackled problem instances. The results of the considered algorithmic approaches show that integer linear programming approaches can only compete with the developed metaheuristics in the context of graphs with up to 100 nodes. When larger graphs are concerned, the application of the populated-based iterated greedy algorithm within the higher-level framework works generally best. The experimental evaluation considers graphs of different types, sizes, densities, and ways of generating the node and edge weights.

1 Introduction

Hard combinatorial optimization problems in which solutions are subsets of the nodes of a given input graph are abundant in the scientific literature. Examples are the maximum independent set (MIS) problem and the minimum dominating set (MDS) problem, as well as node-weighted variants such as, for example, the minimum weight dominating set (MWDS) problem [2].

This work considers the so-called weighted independent domination problem, which is an *NP*-hard combinatorial optimization problem that was initially introduced in [10]. Before describing the problem in technical terms, necessary concepts from graph theory are introduced in the following. Given an undirected graph $G = (V,E)$, V is the set of nodes and E refers to the set of edges. An edge $e \in E$ that connects nodes $u \neq v \in V$ is equally denoted by (u,v) and by (v,u). The *neighborhood* $N(v)$ of a node $v \in V$ is defined as $N(v) := \{u \in V \mid (v,u) \in E\}$, the *closed neighborhood* $N[v]$ of a node $v \in V$ is defined as $N[v] := N(v) \cup \{v\}$, and the set of edges incident to a node $v \in V$ is denoted by $\delta(v)$. Note, in this context, that an edge $e \in E$ is called *incident* to a node v, if v forms one of the two endpoints of e. Given an undirected graph $G = (V,E)$, a subset $D \subseteq V$ of the nodes is called a *dominating set* if every node $v \in V \setminus D$ is adjacent to at least one node from D, that is, if for every node $v \in V \setminus D$ there exists at least one node $u \in D$ such that $v \in N(u)$. Furthermore, a set $I \subseteq V$ is called an *independent set* if for any pair $v \neq v' \in I$ it holds that v and v' are not connected by an edge in G. Moreover, $I \subseteq V$ is called a *maximal independent set* if adding any node from $V \setminus I$ would destroy the independent set property. Note that every maximal independent set is, at the same time, a dominating set. Therefore, a maximal independent set is also called an *independent dominating set*. Vice versa, a subset $D \subseteq V$ is an *independent dominating set* if D is a maximal independent set. Finally, given an independent dominating set $D \in V$, for all $v \in V \setminus D$ we define the *D-restricted neighborhood* $N(v \mid D)$ as $N(v \mid D) := N(v) \cap D$, that is, the neighborhood of v is restricted to all its neighbors that are in D.

1.1 The Weighted Independent Domination Problem

In the weighted independent domination (WID) problem we are given an undirected graph $G = (V,E)$ with node and edge weights. More specifically, for each $v \in V$, respectively $e \in E$, we are given an integer weight $w(v) \geq 0$, respectively $w(e) \geq 0$. The WID problem consists in finding an independent dominating set D in G that minimizes the following cost function:

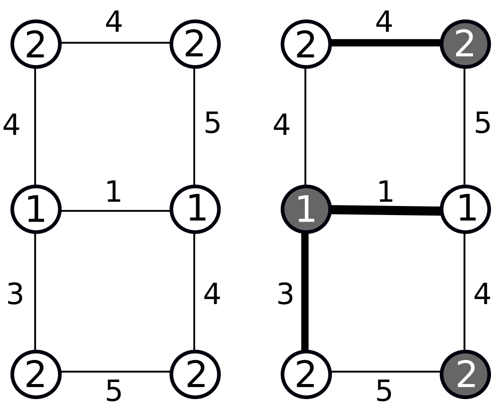
$$f(D) := \sum_{u \in D} w(u) + \sum_{v \in V \setminus D} \min\{w(v,u) \mid u \in N(v \mid D)\} \qquad (1)$$

In words, the objective function value of D is obtained by the sum of the weights of the nodes in D plus the sum of the weights of the minimum-weight edges that connect the nodes that are not in D to nodes that are in D. As an example consider the graphics in Figure 1. The node weights are indicated inside the nodes and the edge weights are provided besides the edges. A possible input graph is shown in Figure 1a, whereas the optimal solution is shown in Figure 1b. The nodes that form part of set D are indicated with a darkgray background. The minimum weight edges that are chosen to connect nodes not in D to nodes in D are indicated with bold lines. The objective function value of the optimal solution is 13, which is composed of the nodes weights (2 + 1 + 2) and the edge weights (4 + 1 + 3).

1.2 Related Problems and Applications

Applications which can be modelled in terms of finding independent and/or dominating sets in graphs are abundant in real life settings. In the following we will give a short overview concerning problems that are closely related to the WID problem.

Maximum indepenent sets According to [14], finding maximum independent sets in undirected graphs has a large variety of applications including coding theory, information retrieval, signal transmission, classification theory, and experimental design, among others. The currently most successful metaheuristic algorithms to solve this problem include the general swap-based multiple neighborhood tabu search proposed in [14] and the fast local search routines presented in [1]. The currently best metaheuristic for the node-weighted variant of the MIS problem, the so-called maximum weight independent set problem, is an iterated local search proposed in [19].



(a) Graph with node and edge weights.

(b) Optimal solution to the WID problem.

Figure 1: Example of the WID problem. (a) shows a possible input graph and (b) shows the corresponding optimal solution.

Minimum dominating sets According to [7], the identification of small dominating sets is important, for example, for misuse detection and gateway placement in wireless adhoc networks, and for query-focused multi-document summarization. Moreover, in [26] the authors point out applications in the deployment of wavelength division multiplexing all-optical networks and in network intrusion detection. Additional applications are mentioned in the context of social networks [25] and for analyzing biological networks [17]. The currently best metaheuristics for solving the minimum weight dominating set problem, which includes the non-weighted MDS problem as a special case, are an ant colony optimization approach and a genetic algorithm from [23], a hybrid evolutionary algorithm from [11], a hybrid approach combining iterated greedy algorithms and an integer linear programming (ILP) solver in a sequential way from [7], a memetic algorithm from [15], and a local search based on configuration checking in [26]. Other recent references include [18, 22].

Minimum independent dominating sets According to [28], finding minimum independent dominating sets has, in particular, applications in the context of clustering approaches and the placement of actors in their respective clusters in wireless networks. Another application in the context of virtual backbone generation in mobile wireless adhoc networks is mentioned in [29]. The most recent metaheuristic approaches for this problem are a greedy randomized adaptive search procedure (GRASP) from [28] and a memetic algorithm from [27].

Note that the WID problem considered in this paper is an extension of the minimum independent dominating set problem by considering (1) node and edge weights, and (2) an augmented objective function that considers the cost of connecting non-selected nodes with selected nodes. Therefore, possible applications of the WID problem may arise in the context of clustering algorithms in (mobile) wireless adhoc networks whenever a cost function is known for connecting non-selected nodes to cluster heads, for example.

1.3 Our Contribution

So far, the WID problem has only been considered from a theoretical perspective. It is easy to see that the problem is *NP*-hard. This is because with $w(v) = 1$ for all $v \in V$ and $w(e) = 0$ for all $e \in E$ it reduces to the independent domination problem which was shown to be *NP*-hard in [9]. A linear time algorithm for the WID problem in series-parallel graphs was proposed in [10]. In this work we consider the WID problem in general graphs from an algorithmic perspective. Our contributions are as follows. First, we present three ILP models for the WID problem. Second, we propose two different greedy heuristics for solving the problem. The first one is known from the minimum weight independent dominating set problem, while the second one is specifically developed for the WID problem. Third, we propose a so-called population-based iterated greedy (PBIG) algorithm. This algorithm employs an iterated greedy metaheuristic in a population-based fashion, and can therefore be seen as a hybrid between methods based on local search and population-based methods. Iterated greedy algorithms have been shown to work very well for many combinatorial optimization problems (see, for example, [24, 12]). The first PBIG approach was proposed in the context of the minimum weight vertex cover problem in [6]. Later, PBIG was also applied to the delimitation and zoning of rural settlements [21] and, as mentioned above, to the minimum weight dominating set problem [7].

Finally, in addition to applying the PBIG algorithm directly to all problem instances, our last contribution consists in applying PBIG within a framework known as *construct, merge, solve & adapt* (CMSA) [5]. Examples for the application of CMSA can be found in [5, 3] for the minimim common string partition problem, and in [4] for the repetition-free longest common subsequence problem. CMSA was initially introduced for being able to take profit from exact solvers—such as, for example, the general purpose ILP solver CPLEX—in the context of problem instances that are too large to be tackled directly by the respective exact solver. CMSA generates, at each iteration, a number of probabilistic solutions which are used to produce sub-instances to the tackled problem instances. The exact solver is then used to solve the corresponding sub-instance at each iteration of CMSA. In fact, in the context of the WID problem we tried to implement a standard CMSA algorithm based on all three ILP models proposed in this work. Unfortunately, these versions of CMSA were still not efficient enough in order to be able to deal with, for example, graphs with 1000 nodes. Therefore, the idea was to study if CMSA can also be used in order to improve the working of a standard metaheuristic such as PBIG. This gave rise to a CMSA-PBIG algorithm which makes use of the framework of CMSA and uses PBIG (instead of an exact solver) for deriving hopefully good solutions to the corresponding sub-instance at each iteration of CMSA. The obtained results show that this is, indeed, the case. Note that this work is a signficiant extension of a paper that appears in the conference proceedings of EvoCOP 2017 [20]. The extension concerns the development of two additional ILP models, the application of PBIG within the CMSA framework, and the application to problem instances that represent different types of graphs.

1.4 Organization

The remainder of this paper is organized as follows. In Section 2, three different ILP models for the WID problem are proposed. Two greedy heuristics are outlined in Section 3. Moreover, the PBIG approach and its application in the CMSA framework are described in Section 4. Finally, an extensive experimental evaluation is provided in Section 5 and conclusions as well as an outlook to future work is given in Section 6.

2 ILP Models

In the following we present three different ILP models for the WIDP problem. These models are experimentally evaluated in Section 5.

2.1 ILP-1: Model based on Indicator Variables

The first one of the proposed ILP models—henceforth called ILP-1—uses three sets of binary variables. For each node $v \in V$ it uses a binary variable x_v. Moreover, for each edge $e \in E$ the model uses a binary variable y_e and a binary variable z_e. Hereby, x_v indicates if v is chosen for the solution. Moreover, z_e indicates if $e \in E$ is selected for connecting a non-chosen node to a chosen one. Variable y_e is an indicator variable, which indicates if e is choosable, or not.

$$
\begin{aligned}
(\text{ILP-1}) \quad \min \quad & \sum_{v \in V} x_v w(v) + \sum_{e \in E} z_e w(e) && && (2)\\
\text{s.t.} \quad & x_v + x_u \leq 1 && \text{for } e = (u,v) \in E && (3)\\
& x_v + \sum_{u \in N(v)} x_u \geq 1 && \text{for } v \in V && (4)\\
& x_v + x_u = y_e && \text{for } e = (u,v) \in E && (5)\\
& z_e \leq y_e && \text{for } e \in E && (6)\\
& x_v + \sum_{e \in \delta(v)} z_e \geq 1 && \text{for } v \in V && (7)\\
& x_v \in \{0,1\} && \text{for } v \in V &&\\
& y_e \in \{0,1\} && \text{for } e \in E &&\\
& z_e \in \{0,1\} && \text{for } e \in E &&
\end{aligned}
$$

Hereby, constraints (3) are the independent set constraints, that is, they make sure that two adjacent nodes can not take part in the solution. Constraints (4) are the dominating set constraints. They ensure that for each node $v \in V$, either the node itself or at least one of its neighbors form part of the solution. These two sets of constraints will be the same in all three ILP models. Constraints (5) ensure the proper setting of the indicator variables. Note that edges that contribute to the objective function value must always connect a node that is not chosen for the solution with a node that is in the solution. Therefore, if—concerning an edge $e = (u,v)$—either v or u is in the solution, variable y_e is forced to take value one, which indicates that this edge is choosable. Constraints (6) relate the indicator variables with the variables that actually show which edges are chosen. In particular, if an indicator variable y_e has value zero, z_e is forced to take value zero, which means e cannot be chosen. Finally, constraints (7) ensure that each node $v \in V$ that does not form part of the solution—that is, when $x_v = 0$—is connected by an edge to a node that forms part of the solution. Due to the fact that the optimization goal concerns minimization, the edge with the lowest weight is chosen for this purpose.

2.2 ILP-2: Eliminating the Indicator Variables

The second ILP model—henceforth called ILP-2—follows the same idea as ILP-1, apart from the fact that it does not require the set of indicator variables. That is, model ILP-2 only makes use of binary variables x_v for all $v \in V$ and binary variables z_e for all $e \in E$. The meaning of these variables is described above.

$$\begin{aligned}
(\textsc{Ilp-2}) \quad \min \quad & \sum_{v \in V} x_v w(v) + \sum_{e \in E} z_e w(e) && && (8)\\
\text{s.t.} \quad & x_v + x_u \leq 1 && \text{for } e = (u,v) \in E && (9)\\
& x_v + \sum_{u \in N(v)} x_u \geq 1 && \text{for } v \in V && (10)\\
& x_v + x_u \geq z_e && \text{for } e = (u,v) \in E && (11)\\
& (1 - x_v) + (1 - x_u) \geq z_e && \text{for } e = (u,v) \in E && (12)\\
& x_v + \sum_{e \in \delta(v)} z_e \geq 1 && \text{for } v \in V && (13)\\
& x_v \in \{0,1\} && \text{for } v \in V &&\\
& z_e \in \{0,1\} && \text{for } e \in E &&
\end{aligned}$$

Note that the independent set constraints (9), the dominating set constraints (10), and constraints (13) that ensure that each node $v \in V$ that does not form part of the solution is connected by an edge to a node that forms part of the solution, are the same as in model ILP-1. However, the two sets of constraints concerning the indicator variables from model ILP-1 (constraints (5) and (6)) are replaced by constraints (11) and (12). Note that when both x_v and x_u—concerning an edge $e = (u,v) \in E$—are set to zero, constraints (11) force variable z_e to take value zero, which means that an edge that connects two non-selected nodes can not be chosen for the solution. Furthermore, when both x_v and x_u—again concerning an edge $e = (u,v) \in E$—are set to one, constraints (17) force variable z_e to take value zero, which means that an edge that connects two selected nodes can not be chosen for the solution.

2.3 ILP-3: Expicit Variables for the Edge-Weight Contribution

The third ILP model—henceforth called ILP-3—is structurally different to ILP-1 and ILP-2. The main idea is to model the edge-weight contribution of each node in terms of an integer variable q_v for all $v \in V$. Obviously, the edge-weight contribution of a selected node $v \in V$—that is, when $x_v = 1$—must be zero, whereas the edge-weight contribution of a non-selected node $v \in V$ must be equal to the weight of the minimum-weight edge that connects this node to a selected node.

$$\begin{aligned}
& (\textsc{Ilp-3}) && &&\\
& \min \sum_{v \in V} x_v w(v) + q_v && && (14)\\
& \text{s.t. } x_v + x_u \leq 1 && \forall\, e = (u,v) \in E && (15)\\
& x_v + \sum_{u \in N(v)} x_u \geq 1 && \forall\, v \in V && (16)\\
& q_v \leq (1 - x_v) M && \forall\, v \in V && (17)\\
& q_v \geq 0 && \forall\, v \in V && (18)\\
& q_v \geq x_u w(e) - \left(x_v M + \sum_{\substack{e'=(v,v') \in \delta(v) \\ \text{s.t. } w(e') < w(e)}} x_{v'} M \right) && \begin{array}{l} \forall\, v \in V, \\ e = (v,u) \in \delta(v) \end{array} && (19)\\
& x_v \in \{0,1\} && \forall\, v \in V &&\\
& q_v \in \{-|V| \cdot M, \ldots, M\} && \forall\, v \in V &&
\end{aligned}$$

Algorithm 1 Greedy Heuristic (GREEDY1)

```
1: input: a undirected graph $G = (V,E)$ with node and edge weights
2: $S := \emptyset$
3: $G' := G$
4: while $V' \neq \emptyset$ do
5:     $v^* := \text{argmax}\{\frac{|N(v|G')|}{w(v)} \mid v \in V'\}$ {Ties are randomly resolved}
6:     $S := S \cup \{v^*\}$
7:     Remove from $G'$ all nodes from $N[v \mid G']$ and their incident edges
8: end while
9: output: An independent dominating set $S$ of $G$
```

Observe that the independent set constraints (15) and the dominating set constraints (16) are again as in ILP-1 and ILP-2. In addition, constraints (17) set the upper bound of an edge-contribution variable to zero in case the corresponding node forms part of the solution, that is, of the independent dominating set. In case a node does not form part of the solution, constraints (17) set the upper bound to a large constant M, which we have set to the maximum weight of all edges of the graph in our implementation. Furthermore, constraints (18) set the lower bound of all edge contributions to zero. Finally, constraints (19) correctly set the lower bound of the edge contributions in order to be equal to the weight of the minimum-weight edge connecting the respective node with one of its selected neighbors.

3 Greedy Heuristics

The first one of two different greedy heuristics developed in this work is a simple extension of a well-known heuristic for the minimum weight independent dominating set problem. Given an input graph G, this heuristic starts with an empty solution $S = \emptyset$ and adds, at each step, exactly one node from the remaining graph $G' = (V', E')$ to S. Initally, the *remaining graph* G' is a copy of G. After adding a node $v \in V'$ to S, all nodes from $N[v \mid G']$—that is, from the closed neighborhood of v in G'—are removed from V'. Moreover all their incident edges are removed from E'. In this way, only those nodes that maintain the property of S being an independent set may be added to S. At each step, the node $v \in V'$ that maximizes $\frac{|N(v|G')|}{w(v)}$ is chosen to be added to S, where $N(v \mid G')$ refers to the neighborhood of v in G'. In other words, nodes with a high degree in the remaining graph G' and with a low node weight are preferred. Note that this greedy heuristic does not take the edge weights into account. They are only considered when calculating the objective function value of the final solution S. The pseude-code of this heuristic, henceforth referred to as GREEDY1, is shown in Algorithm 1.

In contrast to GREEDY1, the second greedy heuristic is designed to take into account the edge weights already during the process of constructing a solution. The algorithmic framework of this greedy heuristic—henceforth denoted by GREEDY2—is the same as the one of GREEDY1. However, the way in which a node is chosen at each step is different. For the description of this greedy heuristic the following notations are required. First, the maximum weight of any edge in E is denoted by $w_{\max}$. Then, let $S \in V$ be a partial solution, that is, S is an independent set which is not yet a dominating set, but which can be extended to be a dominating set. The *auxiliary objective function value* $f^{\text{aux}}(S)$ is defined as $\sum_{v \in V} c(v \mid S)$, where $c(v \mid S)$ is called the *contribution* of node v with respect to partial solution S. Given S, these contributions are defined as follows:

1. If $v \in S$: $c(v \mid S) := w(v)$
2. If $v \notin S$ and $N(v) \cap S = \emptyset$: $c(v \mid S) := w_{\max}$
3. If $v \notin S$ and $N(v) \cap S \neq \emptyset$: $c(v \mid S) := \min\{w(e) \mid e = (v,u), u \in S\}$

Note that in the case of S being a complete solution, it holds that $f(S) = f^{\text{aux}}(S)$. Now, in order to obtain GREEDY2, line 5 of Algorithm 1 must be exchanged with the following one:

$$v^* := \operatorname{argmin}\left\{f^{\text{aux}}(S \cup \{v\}) \mid v \in V'\right\} \tag{20}$$

4 Population-Based Iterated Greedy Algorithm

A high level description of the implemented PBIG approach—henceforth referred to as PBIG—is given in Algorithm 2. Apart from the input graph G, PBIG requires values for five parameters: (1) the population size $p_{\text{size}} \in \mathbb{Z}^+$, (2) the lower bound ($D^l$) and the upper bound ($D^u$) for the degree of destruction applied to each solution of the population at each iteration, (3) the determinism rate $d_{\text{rate}} \in [0,1]$, and (4) the candidate list size $l_{\text{size}} > 0$. The latter two parameters control the greediness of the probabilistic solution (re-)construction procedure. Moreover, note that for the values of the above-mentioned bounds it must hold that $0 \leq D^l \leq D^u \leq 1$. For the following description, each solution S is a subset of the nodes of V, has an objective function value $f(S)$, and an individual, possibly dynamic, destruction rate D_S.

The algorithm works as follows. First, the p_{size} solutions of the initial population are generated by function GenerateInitialPopulation$(p_{\text{size}}, d_{\text{rate}}, l_{\text{size}})$ (see line 2 of Alg. 2). Afterwards, each iteration consists of the following steps. First, an empty population $\mathscr{P}_{\text{new}}$, called offspring population, is created. Then, each solution $S \in \mathscr{P}$ is partially destroyed using procedure DestroyPartially(S) (see line 6 of Alg. 2). This results in a partial solution $\hat{S}$. On the basis of $\hat{S}$, a complete solution S' is then constructed using procedure Reconstruct$(\hat{S}, d_{\text{rate}}, l_{\text{size}})$ (see line 7 of Alg. 2). Then, the destruction rate D_S of solution S is adapted depending on the quality of solution S' in function AdaptDestructionRate(S, S'). Each newly obtained complete solution is stored in $\mathscr{P}_{\text{new}}$. Note that the two phases of destruction and re-construction are applied to all solutions from $\mathscr{P}$ independently of each other. When the iteration is completed, procedure Accept$(\mathscr{P}, \mathscr{P}_{\text{new}})$ selects the best p_{size} solutions from $\mathscr{P} \cup \mathscr{P}_{\text{new}}$ for the population of the next iteration. In the case of two solutions from $\mathscr{P} \cup \mathscr{P}_{\text{new}}$ being equal, the criterion used for tie-breaking is based on the individual destruction rates. More specifically, the solution S with the highest individual destruction rate D_S is preferred over the other one. Finally, the algorithm terminates when a predefined CPU time limit is reached, and the best found solution is returned. The four procedures that form the core of PBIG are described in more detail in the following.

GenerateInitialPopulation$(p_{\text{size}}, d_{\text{rate}}, l_{\text{size}})$: This function generates p_{size} solutions for the initial population. For this purpose it uses the mechanism of GREEDY2[1] (see Section 3) in a probabilistic way. At each construction step, first, a random number $l \in [0,1]$ is generated. In case $l \leq d_{\text{rate}}$, the best node according to the greedy function is chosen. Otherwise, a candidate list of size $\min\{|V'|, l_{\text{size}}\}$, where $V' \subseteq V$ are the nodes that can be selected at the current construction step, is generated, and one of the nodes from the candidate list is chosen uniformly at random. Note also that the initial destruction rate

[1]Note that GREEDY2 is chosen over GREEDY1 because, as it will be shown later, GREEDY2 generally works better than GREEDY1.

Algorithm 2 PBIG for the WID problem

```
 1: input: input graph $G$, parameters $p_{\text{size}} > 0$, $D^l, D^u, d_{\text{rate}}, l_{\text{size}} \in [0,1]$
 2: $\mathscr{P} :=$ GenerateInitialPopulation$(p_{\text{size}}, d_{\text{rate}}, l_{\text{size}})$
 3: while termination condition not satisfied do
 4:     $\mathscr{P}_{\text{new}} := \emptyset$
 5:     for each candidate solution $S \in \mathscr{P}$ do
 6:         $\hat{S} :=$ DestroyPartially$(S)$
 7:         $S' :=$ Reconstruct$(\hat{S}, d_{\text{rate}}, l_{\text{size}})$
 8:         AdaptDestructionRate$(S, S')$
 9:         $\mathscr{P}_{\text{new}} := \mathscr{P}_{\text{new}} \cup \{S'\}$
10:     end for
11:     $\mathscr{P} :=$ Accept$(\mathscr{P}, \mathscr{P}_{\text{new}})$
12: end while
13: output: argmin $\{f(S) \mid S \in \mathscr{P}\}$
```

(D_S) of each solution S is set to the lower bound D^l for the destruction rates.

DestroyPartially(S): In this function, $\max\{3, \lfloor D_S \cdot |S| \rfloor\}$ randomly selected nodes are removed from S, where D_S is the current individual destruction rate of solution S.

Reconstruct$(\hat{S}, d_{\text{rate}}, l_{\text{size}})$: Given as input a partial solution $\hat{S}$, this function re-constructs a complete solution S' in the same way in which solutions are probabilistically constructed in the context of generating the initial population (see above). Moreover, the initial destruction rate $D_{S'}$ of S' is set to D^l.

AdaptDestructionRate(S, S'): The individual destruction rate D_S of solution S (from which partial solution $\hat{S}$ was obtained) is updated on the basis of the lower bound D^l and the upper bound D^u as follows. If $f(S') < f(S)$, the value of D_S is set back to the lower bound D^l. Otherwise, the value of D_S is incremented by a certain amount. After initial experiments, we determined this amount to be 0.05. If the value of D_S, after this update, exceeds the upper bound D^u, it is set back to the lower bound D^l.

Note that the idea behind this way of dynamically changing the value of D_S is as follows. As long as the algorithm is able to improve a solution using a low destruction rate, this rate is kept low. In this way, the re-construction is faster. Only when the algorithm seems not to be able to improve over a solution, the individual destruction rate of this solution is increased in a step-wise manner.

4.1 Application of PBIG in the CMSA Framework

As mentioned before, the CMSA framework was introduced in [5] in order to be able to take profit from an efficient exact solver even in the context of problem instances that are too large to be solved directly by the exact solver. The general idea of CMSA is as follows. At each iteration, solutions to the tackled problem instance are generated in a probabilistic way. The solution components found in these solutions are then added to a sub-instance of the original problem instance. Subsequently, an exact solver such as, for example, CPLEX is used to solve the sub-instance to optimality. Moreover, the algorithm is equipped with a mechanism for deleting seemingly useless solution components from the sub-instance. This is done such that the sub-instance has a moderate size and can be solved rather quickly to optimality.

In the context of the WID problem, the set of solution components corresponds to the set of nodes

Algorithm 3 CMSA-PBIG for the WID problem

```
1: input: input graph $G$, parameter values for PBIG and parameter values for $d^{cmsa}_{\text{rate}}$, $l^{cmsa}_{\text{size}}$, $\text{age}_{\max}$, $n_{\text{a}}$, $t_{\max}$, and $\text{opt}_{\text{greedy}}$
2: $S_{\text{bsf}} := \text{NULL}$
3: $V' := \emptyset$
4: $age[v] := 0$ for all $v \in V$
5: while CPU time limit not reached do
6:     for $i = 1, \ldots, n_{\text{a}}$ do
7:         $S :=$ ProbabilisticSolutionGeneration($\text{opt}_{\text{greedy}}$, $d^{cmsa}_{\text{rate}}$, $l^{cmsa}_{\text{size}}$)
8:         for all $v \in S$ and $v \notin V'$ do
9:             $age[v] := 0$
10:            $V' \leftarrow V' \cup \{v\}$
11:        end for
12:    end for
13:    $S'_{\text{pbig}} \leftarrow$ ApplyPBIG($V'$)
14:    if $f(S'_{\text{pbig}}) < f(S_{\text{bsf}})$ then $S_{\text{bsf}} := S'_{\text{pbig}}$
15:    Adapt($V'$, $S'_{\text{pbig}}$, $\text{age}_{\max}$)
16: end while
17: output: $S_{\text{bsf}}$
```

of the input graph. Moreover, solutions can be probabilistically constructed by a probabilistic version of either GREEDY1 or GREEDY2. In fact, both greedy heuristics can be made probabilistic by the mechanism described in the context of PBIG in the previous section. For this purpose we require two parameters: (1) the determinism rate parameter (called d^{cmsa}_{rate} in the context of CMSA-PBIG) and (2) the candidate list size (called l^{cmsa}_{size} in the context of CMSA-PBIG). Our initial idea was to use one of the three proposed ILP models in order to solve the sub-instances within CMSA-PBIG to optimality. However, even though smaller than the original problem instances, this idea turned out to be inefficient in the case of graphs with 500 and 1000 nodes. Therefore, we implemented the following option. Instead of applying an exact solver to each sub-instance, the PBIG algorithm is applied with a certain time limit to each sub-instance. The resulting pseudo-code of CMSA-PBIG is provided in Algorithm 3.

Each algorithm iteration works as follows. First, the best-so-far solution S_{bsf} is initialized to NULL, indicating that no such solution exists yet. Moreover, the current sub-instance $V' \subseteq V$ (where V is the set of nodes of the input graph G) is initialized to the empty set. Then, at each iteration, n_{a} solutions are probabilistically generated in function ProbabilisticSolutionGeneration($\text{opt}_{\text{greedy}}$, d^{cmsa}_{rate}, l^{cmsa}_{size}), either making use of GREEDY1 (in case $\text{opt}_{\text{greedy}} = 0$) or of GREEDY2 (in case $\text{opt}_{\text{greedy}} = 1$). The nodes found in the constructed solutions are then added to V'. Furthermore, each node $v \in V'$ has an *age*, labelled $age[v]$, which is initialized to zero. Next, PBIG is applied in function ApplyPBIG(V') to find a high-quality solution, however, restricted to the nodes from V'; that is, the solution (re-)construction process of PBIG is restricted to choose nodes from V'. If the resulting solution, labelled S'_{pbig}, is better than the current best-so-far solution S_{bsf}, solution S'_{pbig} is adopted as the new best-so-far solution. Next, sub-instance V' is adapted on the basis of solution S'_{pbig} in conjunction with the age values of the nodes in V'. This is done in function Adapt(V', S'_{pbig}, $\text{age}_{\max}$) as follows. First, the age of each node in $V' \setminus S'_{\text{pbig}}$ is incremented while the age of each node in S'_{pbig} is re-initialized to zero. Subsequently, those nodes from V' with an age value greater than $\text{age}_{\max}$—which is a parameter of the algorithm—are removed from V'. This causes that nodes that are never selected for the best solutions of PBIG do not slow down the working of PBIG in coming iterations.

5 Experimental Evaluation

The following seven algorithmic approaches are evaluated on a large variety of benchmark instances: (1–3) the application of CPLEX to the three ILP models (ILP-1, ILP-2, and ILP-3), (4) GREEDY1, (5) GREEDY2, (6) PBIG, and (7) CMSA-PBIG. All techniques were implemented in ANSI C++ using GCC 4.6.3 for compiling the software.[2] Moreover, we used CPLEX version 12.6 in single-threaded execution for solving the ILP models. The experimental results that are presented in the following were obtained on a cluster of 32 computers with Intel® Xeon® X5660 CPUs of 6 nuclei of 2.8 GHz and (in total) 48 Gigabytes of RAM. For each run of CPLEX we allowed a maximum of 4 Gigabytes of RAM. In the following, first, the set of benchmark instances is described. Then, a detailed analysis of the experimental results is presented.

5.1 Benchmark Instances

Two types of graphs were considered for the experimental evaluation: (1) random graphs and (2) random geometric graphs. In each case, graphs of different properties—for what concerns, for example, the density—and different sizes were created. In particular, for each type we generated graphs of 100, 500 and 1000 nodes, that is, $|V| \in \{100, 500, 1000\}$. The random graphs were generated adding edges between nodes totally at random, with a given probability *ep* for each edge. This probability controls the density of the graph. In particular, we considered $ep \in \{0.05, 0.15, 0.25\}$. The random geometric graphs were created as follows. First, the $|V|$ nodes were assigned to random coordinates from the unit square. Then, a *radius* (r) was fixed and each pair of nodes at a distance smaller or equal than the radius was connected by an edge. The radius controls the density of the graph, that is, the larger the radius the denser is the resulting graph. In order to produce graphs with densities comparable to the ones of the random graphs we considered $r \in \{0.14, 0.24, 0.34\}$. The main difference between random geometric graphs and random graphs is that in the former ones only nodes that are placed close together may be connected while in the latter ones any two nodes may be connected.

Three different schemes for generating the node and edge weights were considered. In the first scheme, both node and edge weights were drawn uniformly at random from $\{0, \ldots, 100\}$. Henceforth, we call the resulting graphs *neutral graphs*. In the second scheme, node weights were drawn uniformly at random from $\{0, \ldots, 1000\}$ and edge weights were drawn uniformly at random from $\{0, \ldots, 10\}$. In these graphs, henceforth called *node oriented* graphs, the choice of the nodes is presumably very important because of the nodes themselfs. Finally, in the third scheme node weights were drawn uniformly at random from $\{0, \ldots, 10\}$ and edge-weights were drawn uniformly at random from $\{0, \ldots, 1000\}$. In these *edge-oriented* graphs, the choice of the nodes is important due to edges that are made available for connecting non-chosen nodes to chosen nodes.

For each combination of a graph type, a number of nodes, an edge probability (respectively, a radius), and a weight generation scheme, we produced 10 problem instances. This makes a total of 540 graphs: 270 random graphs (this set is henceforth called RG) and 270 random geometric graphs (this set is henceforth called RGG). All graphs can be downloaded from `https://www.iiia.csic.es/~christian.blum/research.html`.

[2]Executables of GREEDY1, GREEDY2, PBIG and CMSA-PBIG, compiled for working under Ubuntu Linux, are available from `https://www.iiia.csic.es/~christian.blum/research.html`.

5.2 Tuning Experiments

The automatic configuration tool irace [16] was used in order to find well-working values for the parameters of PBIG and CMSA-PBIG. In this section we describe the experimental setup used for the tuning experiments, and the tuning results.

5.2.1 Tuning of PBIG

The following five parameters were considered in the case of PBIG: p_{size}, D^l, D^u, d_{rate} and l_{size}. The tuning tool was applied separately for each combination of graph type, number of nodes and the weight generation scheme. Note that no separate tuning was performed concerning the graph density (depending on ep in the context of random graphs, respectively r in the context of random geometric graphs). This is because, after initial runs, it was shown that the other parameters have a higher influence on the behavior of the algorithm. Summarizing, irace was applied 18 times with a budget of 1000 applications of PBIG per tuning run.

For each application of PBIG a time limit of $|V| \cdot 3$ CPU seconds was fixed. For each run of irace, two tuning instances were generated for each combination of graph type, number of nodes, graph density, and weight generation scheme. This gives a total of six tuning instances per run of irace. The following parameter value ranges were considered for each tuning run:

- $p_{\text{size}} \in \{1, 10, 50, 100\}$.
- For the lower and upper bound values of the destruction percentage, the following value combinations were considered: $(D^l, D^u) \in$ {(10,10), (20,20), (30,30), (40,40), (50,50), (60,60), (70,70), (80,80), (90,90), (10,50), (30,70), (50,90)}. Note that in those cases in which both bounds have the same value, the percentage of deleted nodes is always the same.
- $d_{\text{rate}} \in \{0.0, 0.3, 0.5, 0.7, 0.9\}$.
- $l_{\text{size}} \in \{1, 3, 5, 10\}$.

The results of the tuning processes in the case of random graphs are presented in Table 1. The trends are as follows: the population size (p_{size}) should be rather high. Interestingly, the option of a dynamically changing value for the destruction rate (D^l, D^u) never resulted best. In most cases a fixed value greater than 0.5 is selected. The determinism rate (d_{rate}) should be rather low, specially when large graphs are concerned. Finally, the candidate list size (l_{size}) should be rather high.

In the same way, the results of the tuning processes for random geometric graphs are shown in Table 2. Here, the trends are as follows: the chosen population sizes are rather high, with one exception ($|V| = 100$, node-oriented). Interestingly, this is also the only case in which a dynamically changing destruction rate was selected. In the other cases a fixed value greater than 0.4 is selected for this parameter. The selected determinism rate tends to decrease with increasing graph size, and the candidate list size should be rather high.

5.2.2 Tuning of CMSA-PBIG

In addition to the five parameters of PBIG, the tuning procedure for CMSA-PBIG must additionally consider the six parameters of the CMSA framework of CMSA-PBIG: d_{rate}^{cmsa}, l_{size}^{cmsa}, age_{max}, n_{a}, t_{max}, and $\text{opt}_{\text{greedy}}$. The parameter value ranges for the five PBIG-parameters were chosen as for the tuning procedure of the stand-alone PBIG. For the six additional parameters of CMSA-PBIG, the value ranges considered were the following:

Table 1: Results of tuning PBIG with irace for random graphs.

Weight scheme	$\lvert V\rvert$	p_{size}	(D^l, D^u)	d_{rate}	l_{size}
	100	50	$(0.7, 0.7)$	0.0	5
neutral	500	100	$(0.5, 0.5)$	0.3	10
	1000	100	$(0.5, 0, 5)$	0.0	10
	100	10	$(0.8, 0.8)$	0.0	10
node-oriented	500	100	$(0.6.0.6)$	0.0	5
	1000	100	$(0.6, 0.6)$	0.3	10
	100	100	$(0.6, 0.6)$	0.0	10
edge-oriented	500	50	$(0.6, 0.6)$	0.0	10
	1000	100	$(0.5, 0.5)$	0.0	10

Table 2: Results of tuning PBIG with irace for random geometrics graphs.

Weight scheme	$\lvert V\rvert$	p_{size}	(D^l, D^u)	d_{rate}	l_{size}
	100	50	$(0.8, 0.8)$	0.3	10
neutral	500	100	$(0.5, 0, 5)$	0.5	5
	1000	100	$(0.4, 0.4)$	0.3	5
	100	1	$(0.5, 0.9)$	0.5	10
node-oriented	500	100	$(0.5, 0.5)$	0.3	10
	1000	50	$(0.6, 0.6)$	0.3	5
	100	100	$(0.8, 0.8)$	0.9	10
edge-oriented	500	50	$(0.4, 0.4)$	0.0	10
	1000	100	$(0.4, 0.4)$	0.0	10

- $d_{\text{rate}}^{cmsa} \in \{0.0, 0.3, 0.5, 0.7, 0.9\}$.
- $l_{\text{size}}^{cmsa} \in \{1, 3, 5, 10, 20\}$.
- $\text{age}_{\text{max}} \in \{1, 3, 5, 10, inf\}$.
- $n_{\text{a}} \in \{1, 10, 30, 50\}$.
- $t_{\text{max}} \in \{1, 2, 5, 10, 50\}$.
- $\text{opt}_{\text{greedy}} \in \{0, 1\}$, where value 0 represents the selection of GREEDY1 for the probabilistic construction of solutions, and value 1 the selection of GREEDY2 for this purpose.

The setup of the tuning processes for CMSA-PBIG was the same as for the ones of PBIG. That is, irace was applied 18 times with a budget of 1000 applications of CMSA-PBIG per tuning run. The time limit of $|V| \cdot 3$ CPU seconds per execution was applied, and the same tuning instances were used.

The results of the tuning processes in the case of random graphs are shown in Table 3. Note that when referring, in the following, to PBIG, we mean the application of PBIG within CMSA-PBIG. The following remarkable trends can be observed: the selected destruction rate of PBIG is generally lower than the one chosen for the stand-alone version of PBIG. The determinism rate of PBIG decreases with increasing graph size. Interestingly, the required candidate list size of PBIG is generally lower than the one of the probabilistic solution construction mechanism of the CMSA framework. Concerning the remaining parameters of the CMSA framework, the following can be observed: the determinism rate

Table 3: Results of tuning CMSA-PBIG with irace for random graphs.

Weight scheme	$\|V\|$	p_{size}	(D^l, D^u)	d_{rate}	l_{size}	d^{cmsa}_{rate}	l^{cmsa}_{size}	age_{max}	n_{a}	t_{max}	$\text{opt}_{\text{greedy}}$
neutral	100	50	(0.3, 0.7)	0.7	3	0.5	3	inf	30	1	1
	500	50	(0.3, 0.3)	0.3	5	0.3	5	inf	30	5	1
	1000	10	(0.4, 0.4)	0.0	5	0.3	20	5	10	5	0
node-oriented	100	1	(0.6, 0.6)	0.7	10	0.7	10	inf	30	1	0
	500	10	(0.3, 0.3)	0.3	10	0.5	20	3	50	10	0
	1000	10	(0.5, 0.5)	0.0	5	0.5	20	10	30	50	1
edge-oriented	100	1	(0.5, 0.9)	0.5	5	0.5	5	inf	30	5	1
	500	100	(0.3, 0.3)	0.0	5	0.3	20	3	50	10	0
	1000	50	(0.4, 0.4)	0.3	10	0.5	10	3	50	50	1

Table 4: Results of tuning CMSA-PBIG with irace for random geometrics graphs.

Weight scheme	$\|V\|$	p_{size}	(D^l, D^u)	d_{rate}	l_{size}	d^{cmsa}_{rate}	l^{cmsa}_{size}	age_{max}	n_{a}	t_{max}	$\text{opt}_{\text{greedy}}$
neutral	100	10	(0.3, 0.7)	0.3	3	0.3	20	10	30	2	1
	500	10	(0.3, 0.7)	0.3	3	0.3	20	10	30	2	1
	1000	100	(0.2, 0.2)	0.3	10	0.5	20	inf	30	50	1
node-oriented	100	10	(0.5, 0.5)	0.7	10	0.5	10	1	50	2	0
	500	1	(0.2, 0.2)	0.7	10	0.3	20	inf	30	5	0
	1000	50	(0.2, 0.2)	0.7	3	0.5	10	5	50	5	0
edge-oriented	100	1	(0.3, 0.3)	0.7	10	0.5	10	5	50	5	1
	500	1	(0.3, 0.3)	0.7	10	0.5	10	5	50	5	1
	1000	50	(0.3, 0.7)	0.0	10	0.5	10	10	1	5	1

of the probabilistic solution construction mechanism of CMSA-PBIG tends to decrease with increasing graph size. The number of solutions constructed per iteration (n_{a}) is around 30. The time limit for the application of PBIG at each iteration (t_{max}) increases with increasing graph size, and finally, the selection of GREEDY1 (value 0 of $\text{opt}_{\text{greedy}}$) is most common for graphs generated according to the node-oriented weight scheme, while GREEDY2 seems preferred for the remaining graphs.

In the same way, the results of the tuning processes concerning random geometric graphs are presented in Table 4. The trends that can be observed in this case are very similar to those already outlined for random graphs.

5.3 Numerical Results

The seven solution approaches were applied exactly once to each problem instance. The computation time limit for the applications of CPLEX, PBIG and CMSA-PBIG was $|V| \cdot 3$ seconds for each graph. The results are presented in numerical form in two tables: Table 5 contains the results for all random graphs and Table 6 contains the results for all random geometric graphs. The two tables have the following format. The first three table columns indicate the number of nodes in the graph ($|V|$), the weight generation scheme, and the graph density in terms of the edge probability (ep) for random graphs and the radius (r) for random geometric graphs. The results of the seven approaches are presented in two columns for each approach. The first one of these columns (with heading **result**) provides, in all seven cases, the average result obtained for the corresponding 10 problem instances. In the case of GREEDY1 and GREEDY2, the second column provides the average computation times (in seconds). In the case of the application of CPLEX to the three ILP models, the second columns provide the average optimality gaps (in %) that correspond to the results shown in the first columns. Finally, in the case of PBIG and CMSA-PBIG, the second column shows the average time at which the best solutions of a run were found. Note that the best result of each table row is shown with gray background. In addition, we applied a statistical significance test to the results of each table row. More specifically, in each table row all approaches were compared to the best-performing approach and the results of those approaches who are statistically equivalent to the best-performing approach are marked by the ★ symbol (significance level of 0.05). The statistical differences have been assessed using the Friedman test and the *p*-values have been corrected for multiple comparison using Finner's procedure [13].

Additionally, we aimed for detecting the differences between the algorithms (if any) for large subsets of the problem instances. First, all the algorithms have been compared simultaneously using the Friedman test. Then, given that in all the cases the test rejects the hypothesis that all the algorithms perform equally, all the pairwise comparisons have been tested using the Nemenyi post-hoc test [13]. The corresponding results are shown in Figures 2 and 3 by means of so-called criticial difference plots. Briefly, each approach is positioned in the segment according to its average ranking concerning the considered subset of instances. Then, the critical difference (CD) is computed for a significance level of 0.05 and the performance of those algorithms that have a difference lower than CD are regarded as equal—that is, no difference of statistical significance can be detected. This is indicated in the graphic by horizontal bars joining the respective algorithms.[3]

The experimental results allow us to make the following observations:

- When considering all instances together—see Figure 2a—CMSA-PBIG is the best-performing algorithm, followed by PBIG. The next group of approaches is composed of the application of CPLEX to the three ILP models. Concerning the order between them, ILP-2 is generally the best-performing one, followed by ILP-3 and then ILP-1. Finally, the worst-performing group of algorithms is composed of the two greedy algorithms, with GREEDY2 outperforming GREEDY1. All differences are statistically significant.

- Concerning the two types of graphs that were studied—that is, random graphs vs. random geometric graphs—essentially no differences can be observed in the relative behaviour of the algorithms. This means that the studied techniques are not affected by local structures that are present in graphs.

- Concerning the comparison between GREEDY1 and GREEDY2 it can be observed that, while GREEDY2 outperforms GREEDY1 when all problem instances are considered together—in the context of node-oriented graphs, GREEDY1 has a better average ranking than GREEDY2 (not statistically significant). Concerning computation time, both algorithms are by far the fastest ones in the comparison. Naturally, the computation time of GREEDY2 is slightly higher than that of GREEDY1.

- Comparing the performance of the three ILP models, we can observe that ILP-2 generally achieves the best performance. In particular, for all considered subsets of instances (concerning density and weight generating scheme) ILP-2 outperforms ILP-1. This was to be expected, as the only difference in the two models is the elimination of a set of variables (also resulting in a change of a subset of the constraints). ILP-2 generally also outperforms ILP-3, with the exception of node-oriented graphs, for which ILP-3 achieves a better ranking than ILP-2. Finally, it can also be observed—when looking at the tables containing the numerical results—that all three ILP models are competitive with CMSA-PBIG and PBIG in the context of instances of 100 nodes.

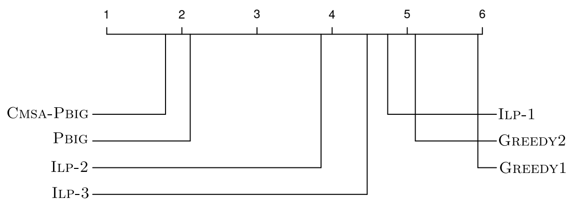
[3]Note that all the tests and the plots have been generated using R's **scmamp** package [8], available at `https://github.com/b0rxa/scmamp`.

Table 5: Numerical results for random graphs.

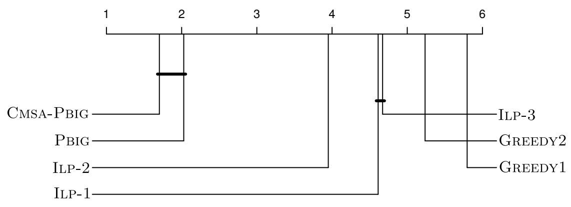
$\|V\|$	Weight scheme	*ep*	GREEDY1		GREEDY2		ILP-1		ILP-2		ILP-3		PBIG		CMSA-PBIG	
			result	**time**	**result**	**time**	**result**	**gap**	**result**	**gap**	**result**	**gap**	**result**	**time**	**result**	**time**
100	N	0.05	3589.1	<0.1	3519.1	<0.1	3049.8	0.7	3049.8	0.0	3051.9★	0.5	3049.8	0.54	3049.8	8.0
		0.15	3014.4	<0.1	2981.3	<0.1	2445.2	30.4	2396.3	28.3	2398.4★	44.6	2330.9	28.31	2338.1★	48.3
		0.25	2883.5	<0.1	2796.1	<0.1	2161.1	31.3	2114.3★	24.3	2167.6	58.8	2070.9	0.16	2093.9★	54.1
	V	0.05	10465.6	<0.1	11756.6	<0.1	7715.4	0.0	7715.4	0.0	7715.4	0.0	7747.0★	76.47	7860.0★	59.4
		0.15	4891.6	<0.1	5845.4	<0.1	3046.6	0.0	3046.6	0.0	3046.6	0.0	3050.3★	16.9	3070.3★	40.6
		0.25	3297.5	<0.1	3488.9	<0.1	1808.4	0.0	1808.4	0.0	1808.4	0.0	1808.4	3.5	1808.4	15.5
	E	0.05	25698.7	<0.1	22269.3	<0.1	14378.7	0.0	14378.7	0.0	14378.7	0.0	14378.7	0.78	14378.7	37.9
		0.15	27528.4	<0.1	23404.5	<0.1	15473.0	42.4	15198.9★	39.6	15098.3★	57.7	14687.8★	0.19	14563.3	17.2
		0.25	25451.4	<0.1	21770.0	<0.1	16001.7	61.5	14557.7★	28.5	15346.3★	69.3	14506.6★	<0.1s	14382.2	37.7
500	N	0.05	14143.1	<0.1	13535.1	<0.1	11857.9	51.9	11809.3	50.5	12882.6	91.7	10327.3★	127.37	10140.6	667.3
		0.15	12268.5	<0.1	11558.0	<0.1	10050.1	69.4	9928.8	68.1	13196.7	98.1	8297.5★	47.33	8046.2	688.2
		0.25	11630.3	<0.1	10429.5	0.1	11341.1	78.0	9465.7★	73.9	12722.4	99.0	7633.7★	10.16	7443.0	538.6
	V	0.05	15501.5	<0.1	18298.1	<0.1	12557.6	52.6	11403.0★	47.1	12059.3	58.0	9822.6★	924.21	9588.2	912.1
		0.15	6496.3	<0.1	7300.1	<0.1	5940.1	61.3	6122.4	59.3	5474.6	80.5	3581.7★	219.52	3557.8	599.8
		0.25	4212.4	<0.1	4463.7	0.1	8166.2	79.4	10145.8	82.9	3628.7★	85.6	2590.6★	52.82	2586.5	521.1
	E	0.05	125357.6	<0.1	108178.0	<0.1	97915.3	82.9	89580.7	81.1	107463.5	98.7	70028.6★	1008.89	67528.9	713.6
		0.15	114951.0	<0.1	102365.1	<0.1	107834.7	93.4	91564.1★	91.7	117036.1	99.9	64673.8★	109.71	62950.1	798.1
		0.25	111012.3	<0.1	99018.2	0.1	100750.3	95.0	88687.7★	94.0	113798.0	99.9	64112.7★	33.26	61411.1	294.9
1000	N	0.05	25569.6	<0.1	23489.7	0.1	25156.1	69.5	25986.0	68.7	27158.7	97.0	17723.7	1750.12	17819.4★	1262.1
		0.15	20827.1	<0.1	20689.1	0.2	21117.8	81.1	18282.9★	76.5	22984.5	99.1	14731.3★	260.52	14461.9	813.3
		0.25	20858.8	<0.1	19280.5	0.4	158097.1	93.2	88053.8	88.7	21821.5	99.5	13968.5★	340.14	13695.6	1480.6
	V	0.05	18048.6	<0.1	20142.3	0.1	35766.3	83.3	39356.1	83.3	15464.1★	77.4	11301.7★	2018.01	11034.6	1363.5
		0.15	7408.3	<0.1	7987.4	0.2	35678.2	91.3	35904.3	97.0	11586.3	92.3	4540.3★	695.71	4456.4	1362.7
		0.25	4941.9★	<0.1	5566.6	0.4	35838.9	96.5	22569.5	100.0	11674.7	96.7	3519.0★	306.67	3460.4	1049.3
	E	0.05	238600.0	<0.1	202992.0	0.1	209391.8	91.1	198540.7	90.1	229778.7	99.7	133667.8★	2121.82	130889.2	1786.2
		0.15	209709.3	<0.1	182726.6	0.2	832437.1	97.7	191911.2	96.2	229888.1	100.0	123760.9★	99.67	120997.2	1478.6
		0.25	198537.0	<0.1	181150.0	0.4	1128240.1	98.6	2041102.4	99.7	221492.0	100.0	124193.3★	447.95	120288.7	1359.1

Table 6: Numerical results for random geometric graphs.

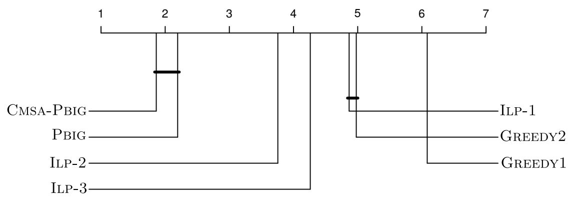
$\|V\|$	Weight scheme	r	GREEDY1		GREEDY2		ILP-1		ILP-2		ILP-3		PBIG		CMSA-PBIG	
			result	**time**	**result**	**time**	**result**	**gap**	**result**	**gap**	**result**	**gap**	**result**	**time**	**result**	**time**
100	N	0.14	3870.6	<0.1	3646.4	<0.1	3261.1	0.0	3261.1	0.0	3261.1	0.0	3261.1	11.93	3261.1	3.3
		0.24	3798.8	<0.1	3378.1	<0.1	2942.9	21.7	2917.5★	15.6	2884.5★	3.0	2882.5	3.04	2882.5	2.7
		0.34	3766.6	<0.1	3388.1	<0.1	2878.5★	26.5	2841.8★	8.6	2876.7★	17.6	2828.0	0.7	2828.0	27.9
	V	0.14	7364.9	<0.1	7514.3	<0.1	5731.8	0.0	5731.8	0.0	5731.8	0.0	5731.8	12.43	5740.0★	2.0
		0.24	2880.1	<0.1	2724.2	<0.1	1981.8	0.0	1981.8	0.0	1981.8	0.0	1981.8	<0.1s	1981.8	2.2
		0.34	1741.0	<0.1	1832.3	<0.1	940.5	0.3	940.5	0.0	940.5	0.0	968.8★	<0.1s	940.5	1.2
	E	0.14	29011.6	<0.1	24998.3	<0.1	19179.4	0.0	19179.4	0.0	19179.4	0.0	19313.2★	117.61	19179.4	10.1
		0.24	35312.1	<0.1	28647.1	<0.1	22519.7	20.5	22325.5★	16.5	22065.9	8.2	22108.3★	6.36	22065.9	5.5
		0.34	37929.4	<0.1	30503.4	<0.1	24295.9★	31.1	23717.6	7.0	24009.2★	19.0	23900.0★	1.14	23717.6	76.7
500	N	0.14	18408.3	<0.1	16208.4	<0.1	14942.7	56.0	15016.4	54.9	15457.2	85.1	13341.5★	835.38	13301.2	548.6
		0.24	18548.8	<0.1	15882.9	<0.1	19028.3	72.7	15668.7	66.5	16788.4	95.5	12943.1★	195.71	12783.3	129.6
		0.34	18311.4	<0.1	15497.8	0.1	18602.1	75.8	15468.3	67.8	18948.1	97.6	13065.5★	6.18	12954.8	327.5
	V	0.14	6807.8	<0.1	7087.8	<0.1	4377.0	0.4	4377.0	0.0	4381.6★	1.4	4400.9★	513.73	4389.7★	165.8
		0.24	3526.6	<0.1	3121.1	<0.1	2619.3★	7.6	2596.4★	5.8	2665.1	39.0	2573.7	9.71	2573.7	17.8
		0.34	2632.2	<0.1	2830.7	0.1	3933.7	51.2	2205.3★	17.0	2305.3	63.8	2181.6	7.17	2181.6	13.1
	E	0.14	177816.7	<0.1	148267	<0.1	128902.8	66.2	128784.8	65.7	129521.2	91.8	112404.8★	730.98	111638.2	680.5
		0.24	181739.2	<0.1	149980.2	<0.1	175979.6	76.6	147009.0	71.5	162967.5	98.3	118765.6★	16.28	117439.8	405.9
		0.34	190996.4	<0.1	155128.6	0.1	180632.5	78.4	149305.9	71.2	188131.4	99.0	122977.3★	5.21	120883.7	346.4
1000	N	0.14	36214.6	<0.1	32393.4★	0.1	38289.8	73.1	33123.7	68.5	38904.3	95.8	25892.9★	776.73	25719.0	1695.2
		0.24	36750.4	<0.1	31462.5	0.2	61533.9	86.9	33744.7	79.7	38610.1	98.3	25570.6★	524.78	25138.9	1081.1
		0.34	36913.2	<0.1	30752.1★	0.3	121566.1	100.0	48329.9	95.9	38549.2	98.9	25714.2★	17.24	25345.0	1698.0
	V	0.14	8552.3	<0.1	8613.7	0.1	6071.9	11.6	6614.5★	14.1	6144.7	43.9	5869.0	1778.52	5890.4★	950.4
		0.24	4977.4	<0.1	5146.2	0.2	11493.0	64.6	12485.5	92.3	4528.1★	74.7	4281.2	109.08	4283.6★	90.1
		0.34	4656.5	<0.1	4693.1	0.4	17662.5	100.0	9742.4	100.0	6214.7	88.7	3974.5★	63.35	3974.3	264.2
	E	0.14	358550.4	<0.1	305034.7	0.1	362985.2	78.8	311510.5	75.5	350946.2	98.4	236226.6★	2582.61	233127.8	1711.2
		0.24	357735.8	<0.1	295099.4 ★	0.2	347041.7	88.6	326670.5	83.2	375956.1	99.4	239560.7★	451.8	238364.5	1277.3
		0.34	369051.9	<0.1	303712.9★	0.3	1232171.4	99.7	476998.4	97.3	370791.1	99.6	244685.7	24.31	246171.8★	1195.2



(a) All instances together.



(b) Random graph instances (set RG).



(c) Random geometric graph instances (set RGG).

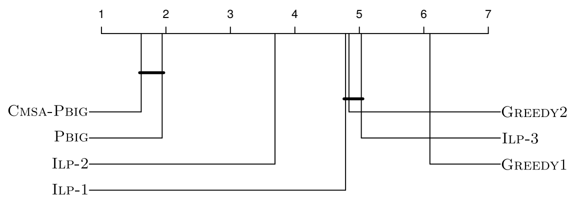
Figure 2: Criticial difference plots for all 540 problem instances together (a), the 270 instances of set RG (b), and the 270 instances of set RGG. The axis shows the average ranking of the seven considered techniques concerning the considered (sub)sets of instances. Horizontal bars connect techniques for which no statistical differences were found.

- CMSA-PBIG and PBIG are clearly (and with statistical significance) the best-performing approaches in our set of compared approaches. This holds when considering all instances together, but also for all subsets of studied instances (see Figures 2 and 3). Concerning the comparison between the two, it can be observered that CMSA-PBIG has—for all considered subsets of instances—a better average ranking than PBIG. This difference is statistically signficiant when considering all instances together, and in the case of edge-oriented graphs. This relative performance between CMSA-PBIG and PBIG is of *general interest*, because it shows that by applying a metaheuristic in a framework such as CMSA, it is possible to improve the performance of the metaheuristic.

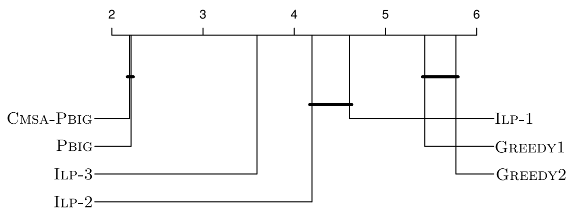
Summarizing, we can state that—in the context of the WID problem—the studied metaheuristics outperform the ILP-based approaches, which in turn outperform the greedy approaches. The ILP-based approaches can only compete with the metaheuristics in the case of instances of 100 nodes. Moreover, we would like to point out again the fact our results have shown it to be possible to improve the performance of a metaheuristic by repeatedly applying it to intelligently generated sub-instances of the tackled problem instances, as it is done in the CMSA framework.

6 Conclusions and Future Work

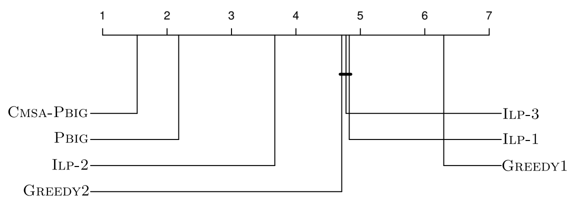
This paper has dealt with an NP-hard problem in graphs, the so-called weighted independent domination problem. We proposed three different integer linear programming models for this problem. Additionally, two different greedy heuristics were presented. The first one of these heuristics is an adaptation of a known heuristic from a related problem to the weighted independent domination problem. This heuristic disregards the edge-weights during the solution construction process. The second heuristic was especially developed for the tackled problem. Finally, we presented a population-based iterated greedy algorithm which takes profit from the better one of the two greedy heuristics. In addition to a standalone application of the population-based iterated greedy algorithm, the algorithm



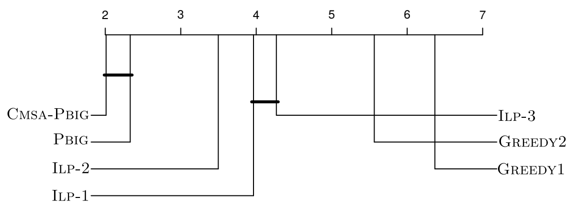
(a) All instances with *neutral* weighting scheme.



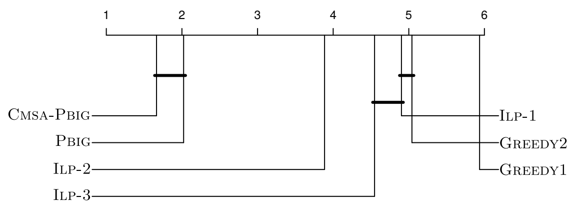
(b) All instances with *node-oriented* weighting scheme.



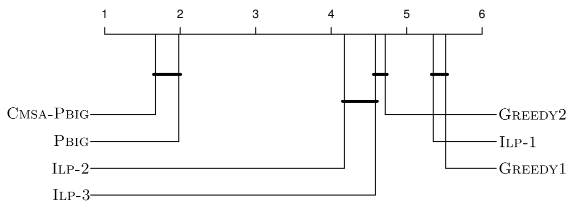
(c) All instances with *edge-oriented* weighting scheme.



(d) All low density instances.



(e) All medium density instances.



(f) All high density instances.

Figure 3: Criticial difference plots for six different subsets of problem instances concerning their weighting scheme and their density. The axis shows the average ranking of the seven considered techniques concerning the considered (sub)sets of instances. Horizontal bars connect techniques for which no statistical differences were found.

was applied within the construct, merge, solve & adapt framework. The results have shown that the population-based iterated greedy algorithm applied within the before-mentioned framework is, with statistical significance, the best-performing approach in the comparison. This is especially interesting, because it shows that the performance of a standard metaheuristic may be improved by the application within a framework such as construct, merge, solve & adapt, which is based on reducing the size of the tackled problem instances.

In the near future we plan to investigate the application of other metaheuristics within the construct, merge, solve & adapt framework in the context of a diverse set of difficult combinatorial optimization problems.

Acknowledgements

This work was supported by project TIN2012-37930-C02-02 (Spanish Ministry for Economy and Competitiveness, FEDER funds from the European Union).

References

[1] Diogo V. Andrade, Mauricio G. C. Resende, and Renato F. Werneck. Fast local search for the maximum independent set problem. *Journal of Heuristics*, 18(4):525–547, 2012.

[2] Rangaswami Balakrishnan and Kanna Ranganathan. *A textbook of graph theory*. Springer Science & Business Media, 2012.

[3] C. Blum. Construct, merge, solve and adapt: Application to unbalanced minimum common string partition. In M. J. Blesa, C. Blum, A. Cangelosi, V. Cutello, A. Di Nuovo, M. Pavone, and E.-G. Talbi, editors, *Proceedings of HM 2016 – 10th International Workshop on Hybrid Metaheuristics*, volume 9668 of *Lecture Notes in Computer Science*, pages 17–31. Springer Verlag, Berlin, Germany, 2016.

[4] C. Blum and M. J. Blesa. Construct, merge, solve & adapt: Application to the repetition-free longest common subsequence problem. In F. Chicano and B. Hu, editors, *Proceedings of EvoCOP 2016 – 16th European Conference on Evolutionary Computation in Combinatorial Optimization*, volume 9595 of *Lecture Notes in Computer Science*, pages 46–57. Springer Verlag, Berlin, Germany, 2016.

[5] Christian Blum, Pedro Pinacho, Manuel López-Ibáñez, and José A. Lozano. Construct, merge, solve & adapt: A new general algorithm for combinatorial optimization. *Computers & Operations Research*, 68:75–88, 2016.

[6] S. Bouamama, C. Blum, and A. Boukerram. A population-based iterated greedy algorithm for the minimum weight vertex cover problem. *Applied Soft Computing*, 12(6):1632 – 1639, 2012.

[7] Salim Bouamama and Christian Blum. A hybrid algorithmic model for the minimum weight dominating set problem. *Simulation Modelling Practice and Theory*, 64:57–68, 2016.

[8] B. Calvo and G. Santafé. scmamp: Statistical comparison of multiple algorithms in multiple problems. *The R Journal*, 8(1), 2016.

[9] G. J. Chang. The weighted independent domination problem is NP-complete for chordal graphs. *Discrete Applied Mathematics*, 143(1):351–352, 2004.

[10] S.-C. Chang, J.-J. Liu, and Y.-L. Wang. The weighted independent domination problem in series-parallel graphs. In *Proceedings of ICS 2014 – The International Computer Symposium*, volume 274 of *Intelligent Systems and Applications*, pages 77–84. IOS Press, 2015.

[11] Sachchida Nand Chaurasia and Alok Singh. A hybrid evolutionary algorithm with guided mutation for minimum weight dominating set. *Applied Intelligence*, 43(3):512–529, 2015.

[12] L. Fanjul-Peyro and R. Ruiz. Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 207(1):55–69, 2010.

[13] S. García and F. Herrera. An extension on “statistical comparisons of classifiers over multiple data sets” for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677 – 2694, 2008.

[14] Yan Jin and Jin-Kao Hao. General swap-based multiple neighborhood tabu search for the maximum independent set problem. *Engineering Applications of Artificial Intelligence*, 37:20 – 33, 2015.

[15] Geng Lin, Wenxing Zhu, and Montaz M Ali. An effective hybrid memetic algorithm for the minimum weight dominating set problem. *IEEE Transactions on Evolutionary Computation*, 20(6):892–907, 2016.

[16] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[17] J. C. Nacher and T. Akutsu. Minimum dominating set-based methods for analyzing biological networks. *Methods*, 102:57–63, 2016.

[18] C. G. Nitash and A. Singh. An artificial bee colony algorithm for minimum weight dominating set. In *Proceedings of SIS 2014 – IEEE Symposium on Swarm Intelligence*, pages 1–7. IEEE press, 2014.

[19] Bruno Nogueira, Rian G. S. Pinheiro, and Anand Subramanian. A hybrid iterated local search heuristic for the maximum weight independent set problem. *Optimization Letters*, 2017.

[20] P. Pinacho Davidson, C. Blum, and J. A. Lozano. The weighted independent domination problem: ILP model and algorithmic approaches. In B. Hu and M. López-Ibáñez, editors, *Proceedings of EvoCOP 2017 – 17th European Conference on Evolutionary Computation in Combinatorial Optimisation*, volume 10197 of *Lecture Notes in Computer Science*, pages 201–214. Springer Verlag, Berlin, Germany, 2017.

[21] Juan Porta, Jorge Parapar, Ramon Doallo, Vasco Barbosa, Inés Santé, Rafael Crecente, and Carlos Díaz. A population-based iterated greedy algorithm for the delimitation and zoning of rural settlements. *Computers, Environment and Urban Systems*, 39:12–26, 2013.

[22] Anupama Potluri and Atul Negi. Some observations on algorithms for computing minimum independent dominating set. In *Proceedings of IC3 2011 – International Conference on Contemporary Computing*, pages 57–68. Springer, 2011.

[23] Anupama Potluri and Alok Singh. Hybrid metaheuristic algorithms for minimum weight dominating set. *Applied Soft Computing*, 13(1):76–88, 2013.

[24] R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.

[25] Peng Gang Sun and Xiaoke Ma. Dominating communities for hierarchical control of complex networks. *Information Sciences*, 414:247–259, 2017.

[26] Yiyuan Wang, Shaowei Cai, and Minghao Yin. Local search for minimum weight dominating set with two-level configuration checking and frequency based scoring function. *Journal of Artificial Intelligence Research*, 58:267–295, 2017.

[27] Yiyuan Wang, Jiejiang Chen, Huanyao Sun, and Minghao Yin. A memetic algorithm for minimum independent dominating set problem. *Neural Computing and Applications*, 2017. In press.

[28] Yiyuan Wang, Ruizhi Li, Yupeng Zhou, and Minghao Yin. A path cost-based GRASP for minimum independent dominating set problem. *Neural Computing and Applications*, 2016. In press.

[29] Xin-Fang Yan, Ai-Qin Liu, and Ting Yang. A virtual backbone network algorithm based on a minimal independent dominating set for manets. *Acta Electronica Sinica*, 35(6):1134–1138, 2007.