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 theoretical work from the literature, this paper considers the problem from an algorithmic perspective. The first contribution consists in the development of an integer linear programming model and a heuristic that makes use of this model. Second, two greedy heuristics are proposed. Finally, the last contribution is a population-based iterated greedy algorithm that takes profit from the better one of the two developed greedy heuristics. The results of the compared algorithmic approaches show that small problem instances based on random graphs are best solved by an efficient integer linear programming solver such as CPLEX. Larger problem instances are best tackled by the population-based iterated greedy algorithm. The experimental evaluation considers random graphs of different sizes, densities, and ways of generating the node and edge weights.

1 Introduction

The so-called weighted independent domination (WID) problem is a combinatorial optimization problem that was introduced in [4]. The problem is an extension of the well-known independent domination (ID) problem. 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 defined as

$\delta(v) := \{e = (v, u) \in E\}$. 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 $v \in V \setminus D$ 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 no two nodes from I are adjacent to each other. Correspondingly, a subset $D \subseteq V$ is called an *independent dominating set* if D is both an independent set and a dominating 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.

In the 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:

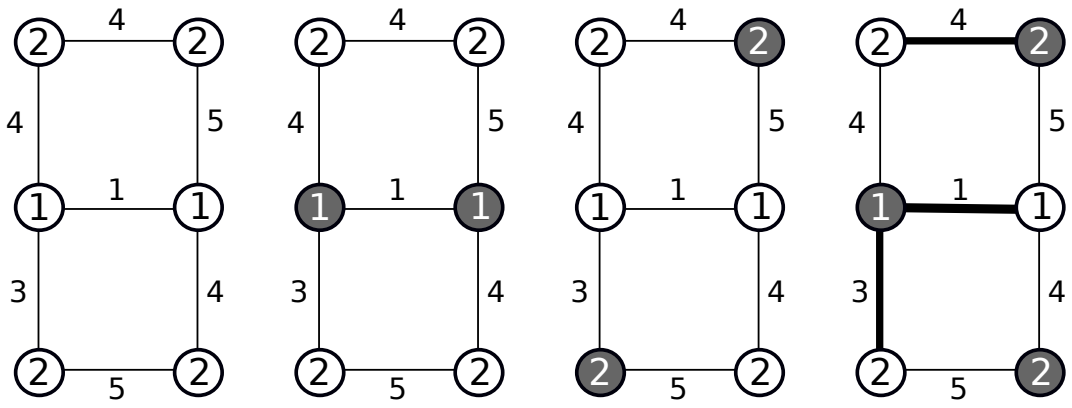
$$f(D) := \sum_{u \in D} w(u) + \sum_{v \in V \setminus D} \min\{w(v, u) \mid u \in N(v \mid D)\} \tag{1}$$

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. An optimal *minimum weight dominating set* (the set of gray nodes) is shown in Figure 1b. However, note that this set is not an independent set because the two nodes that form the set are adjacent to each other. An optimal *minimum weight independent dominating set*[1] is given in Figure 1c. Note that for both, the minimum weight dominating set problem and the minimum weight independent dominating set problem, the edge weights are not considered. Finally, the optimal solution to the WID problem is shown in Figure 1d. 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 this solution is 13, which is composed of the nodes weights (2 + 1 + 2) and the edge weights (4 + 1 + 3).

1.1 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 $e \in E$ the problem reduces to the independent domination problem which was shown to be NP-hard in [3]. A linear time algorithm for the WID problem in series-parallel graphs was proposed in [4]. In this work we consider the WID problem in general graphs from an algorithmic perspective. Our contributions are as follows. First, we present an integer linear programming (ILP) model for the WID problem, together with an ILP-based heuristic. 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

[1] In this problem, given an undirected graph with node weights, the goal is to find an independent dominating set for which the sum of the weights of the nodes is minimal.



(a) Graph with node and edge weights.

(b) Minimum weight dominating set.

(c) Minimum weight dominating independent set.

(d) Optimal solution to the WID problem.

Figure 1: Example that relates the WID problem with the *minimum weight dominating set* problem and with the *minimum weight independent dominating set* problem.

specifically developed for the WID problem. Finally, 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.

1.2 Related Work

On one side, there is related work for problems similar to the one considered in this work. The *minimum independent dominating set* problem, for example, has recently been tackled by a greedy randomized adaptive search procedure (GRASP) in [12]. Another related problem is the *minimum weight dominating set* problem. This problem has been quite popular in recent years as a test case for metaheuristics. The most recent research efforts for this problem have led to the development of an ant colony optimization approach and a genetic algorithm in [10], a hybrid evolutionary algorithm in [5], a hybrid approach combining iterated greedy algorithms and an ILP solver in a sequential way in [2], and a memetic algorithm in [7].

On the other side, there is related work concerning the employed optimization technique, that is, PBIG. In general, iterated greedy (IG) algorithms have shown to be able to work very well in the context of problems for which a good and fast greedy heuristic is known. Prime examples include those to various scheduling problems such as [11, 6]. The first PBIG approach was proposed in the context of the minimum weight vertex cover problem in [1]. Later, PBIG was also applied to the delimitation and zoning of rural settlements [9] and, as mentioned above, to the minimum weight dominating set problem [2].

1.3 Organization

The remainder of this paper is organized as follows. In Section 2 an ILP model for the WID problem is proposed. The greedy heuristics are outlined in Section 3, the ILP-based heuristic is presented in Section 4, and the PBIG approach is described in Section 5. Finally, an extensive experimental evaluation is provided in Section 6 and conclusions as well as an

outlook to future work is given in Section 7.

2 An ILP Model

The proposed ILP 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.

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

Hereby, constraints (3) are the independent set constraints, that is, they make sure that no two adjecent nodes can form part of the solution. Constraints (4) 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 (5) 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. Constraints (6) 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 must form part of the solution. 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.

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' to S. Initally, the *remaining graph* G' is a

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: $\quad v^* := \text{argmax}\{\frac{|N(v|G')|}{w(v)} \mid v \in V'\}$
6: $\quad S := S \cup \{v^*\}$
7: $\quad$ 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

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:

$$v^* := \text{argmin}\left\{f^{\text{aux}}(S \cup \{v\}) \mid v \in V'\right\} \tag{8}$$

4 Heuristic Based on the ILP Model

One possiblity to take profit from the ILP model outlined in Section 2 is to devise a heuristic based on graph reduction. The main idea is to remove a certain percentage of the edges with

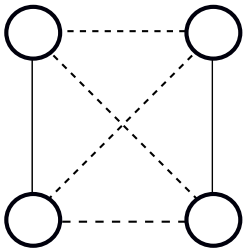


Figure 2: Example for graph reduction. The edge set E of the input graph G consists of all dashed and continuous lines. The edge set E' of the reduced graph G' only consists of the continous lines.

the highest weights from the input graph G, which results in a reduced graph G'. Then, a general-purpose ILP solver such as CPLEX is used to solve the problem in G', forcing that the provided solution is also a feasible solution for G. However, this is not trivial, as indicated by the example in Figure 2. In this example, the edge set E of the input graph G consists of all dashed and continuous lines. The edge set E' of the reduced graph G' only consists of the continous lines. A feasible solution in the original graph consists of exactly one of the four nodes. As a consequence, the remaining three nodes must be connected to the chosen node. Observe that none of these solutions can be generated in the reduced graph. Therefore, for solving the problem in G' we devised the following ILP model, which makes use of additional binary variables p_v for all $v \in V$. Moreover, let w_v denote the weight of the edge with the highest weight of all those edges incident to v, that is, $w_v := \max\{w(e) \mid e \in \delta(v)\}$.

$$\text{(ILP2)} \quad \min \sum_{v \in V} (x_v w(v) + p_v w_v) + \sum_{e \in E'} z_e w(e) \tag{9}$$

$$\text{s.t.} \quad x_v + x_u \leq 1 \qquad \text{for } e = (u,v) \in E \tag{10}$$

$$x_v + x_u = y_e \qquad \text{for } e = (u,v) \in E' \tag{11}$$

$$z_e \leq y_e \qquad \text{for } e \in E' \tag{12}$$

$$x_v + \sum_{u \in N(v)} x_u \geq 1 \qquad \text{for } v \in V \tag{13}$$

$$x_v + p_v + \sum_{e \in \delta'(v)} z_e \geq 1 \qquad \text{for } v \in V \tag{14}$$

$$x_v, p_v \in \{0,1\} \qquad \text{for } v \in V$$

$$y_e \in \{0,1\} \qquad \text{for } e \in E'$$

$$z_e \in \{0,1\} \qquad \text{for } e \in E'$$

Note that constraints (10) and the neighborhood function N in constraints (13) are defined using input graph G. This is done such that the set of nodes chosen in any solution form a valid solution for the original input graph G. In contrast, constraints (11), (12) and the incidence function $\delta'()$ of constraints (14) refer to the edge set E' of the reduced graph G'. This is because for a solution of ILP2 only edges of the reduced graph may be chosen. In comparison to the original ILP, ILP2 has an objective function which is augmented by the term $\sum_{v \in V} p_v \cdot w_v$ and the left-hand-side of constraints (14) is augmented by summing p_v. This has the effect that, in those cases in which any feasible solution for G causes that node v

cannot be connected to any chosen node using an edge from E', variable p_v is forced to take value one. This, in turn, results in summing the weight of the highest-weight edge from E which is incident to v to the objective function value.

In summary, the ILP-based heuristic—henceforth called ILP-HEURISTIC—works as follows. First, heuristic GREEDY2 is applied to G. Second, graph G is reduced by removing $X\%$ of the highest-weight edges, without removing any edges used by the solution of GREEDY2 and without removing more than $(100 - X)\%$ of the edges incident to any node in G. This is done by ordering all edges in E according to decreasing edge weight, and considering one edge after the other for removal, from left to right. This process results in a graph G'. Then, CPLEX is applied to G' using model ILP2. Moreover, the solution of GREEDY2 is provided as a warm-start to CPLEX. This process results in a set S' of chosen nodes. On the basis of S' we generate the corresponding solution in G by simply connecting any node in $V \setminus S'$ using the edge from E with the lowest weight to any of the nodes in S'. Note that by preventing any edges used in the GREEDY2 solution from being removed from G during the graph reduction step, the solution provided by ILP-HEURISTIC must always be at least as good as the one provided by GREEDY2.

5 PBIG: Population-Based Iterated Greedy

A high level description of the implemented PBIG approach 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 $\mathcal{P}_{\text{new}}$, called offspring population, is created. Then, each solution $S \in \mathcal{P}$ 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 $\mathcal{P}_{\text{new}}$. Note that the two phases of destruction and re-construction are applied to all solutions from $\mathcal{P}$ independently of each other. When the iteration is completed, procedure Accept$(\mathcal{P}, \mathcal{P}_{\text{new}})$ selects the best p_{size} solutions from $\mathcal{P} \cup \mathcal{P}_{\text{new}}$ for the population of the next iteration. In the case of two solutions from $\mathcal{P} \cup \mathcal{P}_{\text{new}}$ 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 procedures that form the core of PBIG are described

Algorithm 2 PBIG for the WID problem

input: input graph G, parameters $p_{\text{size}} > 0$, $D^l, D^u, d_{\text{rate}}, l_{\text{size}} \in [0,1]$
$\mathcal{P} :=$ GenerateInitialPopulation$(p_{\text{size}}, d_{\text{rate}}, l_{\text{size}})$
while termination condition not satisfied **do**
  $\mathcal{P}_{\text{new}} := \emptyset$
  for each candidate solution $S \in \mathcal{P}$ **do**
    $\hat{S} :=$ DestroyPartially(S)
    $S' :=$ Reconstruct$(\hat{S}, d_{\text{rate}}, l_{\text{size}})$
    AdaptDestructionRate(S, S')
    $\mathcal{P}_{\text{new}} := \mathcal{P}_{\text{new}} \cup \{S'\}$
  end for
  $\mathcal{P} :=$ Accept$(\mathcal{P}, \mathcal{P}_{\text{new}})$
end while
output: argmin $\{f(S) \mid S \in \mathcal{P}\}$

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[2] (see Section 3) in a probabilistic way. At each construction step, first, a random number $\delta \in [0,1]$ is generated. In case $\delta \leq d_{\text{rate}}$, 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 (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

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

be able to improve over a solution, the individual destruction rate of this solution is increased in a step-wise manner.

6 Experimental Evaluation

The following five algorithmic approaches are evaluated on a variety of benchmark instances: (1) GREEDY1, (2) GREEDY2, considering edge-weights during the solution construction, (3) the application of the ILP solver CPLEX to the ILP model presented in Section 2 (CPLEX), (4) the ILP-based heuristic (ILP-HEURISTIC), and (5) PBIG. All techniques were implemented in ANSI C++ using GCC 4.6.3 for compiling the software. Moreover, we used CPLEX version 12.6 in single-threaded execution. The experimental results that are presented in the following were obtained on a cluster of computers with Intel® Xeon® CPU 5670 CPUs of 12 nuclei of 2933 MHz and (in total) 32 Gigabytes of RAM. For each run of CPLEX we allowed a maximum of 2 Gigabytes of RAM, which was never reached within the alloted computation time. In the following, first, the set of benchmark instances is described. Then, a detailed analysis of the experimental results is presented.

6.1 Benchmark Instances

For the evaluation of the proposed algorithms we used random graphs of various sizes and densities. In particular, we generated graphs of 100, 500 and 1000 nodes, that is, $|V| \in \{100, 500, 1000\}$. Edges between nodes were generated totally at random, with a given probability *ep* for each edge. This probability controls the density of the graph. In particular, we considered $ep \in \{0.05, 0.15, 0.25\}$. 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 high weights associated to the nodes. 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 graph size, edge probability, and weight generation scheme we produced 10 problem instances. This makes a total of 270 graphs.

6.2 Tuning of PBIG

The five concerned parameters are the following ones: p_{size}, D^l, D^u, d_{rate} and l_{size}. The automatic configuration tool irace [8] was applied separately for each combination of the number of nodes and the weight generation scheme. Note that no separate tuning was performed concerning the graph density (depending on *ep*). 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 9 times with a budget of 1000 applications of PBIG per tuning run.

For each application of PBIG a time limit of $|V|/5$ CPU seconds was given. For each run of irace, two tuning instance were generated for each combination of number of nodes, graph

Table 1: Results of tuning PBIG with irace.

Weight scheme	$\|V\|$	p_{size}	(D^l, D^u)	d_{rate}	l_{size}
neutral	100	100	$(0.7, 0.7)$	0.3	10
	500	100	$(0.5, 0.9)$	0.0	10
	1000	100	$(0.4, 0.4)$	0.3	10
node-oriented	100	50	$(0.5, 0.5)$	0.3	10
	500	50	$(0.6, 0.6)$	0.5	10
	1000	100	$(0.5, 0.5)$	0.3	10
edge-oriented	100	100	$(0.5, 0.5)$	0.0	5
	500	100	$(0.5, 0.5)$	0.0	10
	1000	100	$(0.4, 0.4)$	0.0	10

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 are shown in Table 1. The trends are very clear. The population size should be rather high, the determinism rate rather low, and the candidate list size rather high. Moreover, a dynamically changing value of the destruction rate does not seem to be necessary. In most cases a fixed value of around 0.5 is selected.

6.3 Numerical Results

The results are presented in numerical form in Table 2, which has 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). The results of GREEDY1, GREEDY2 and ILP-HEURISTIC are presented by means of two columns each. The first column presents in each row the average result obtained for the corresponding 10 problem instances. The second column provides the average computation times (in seconds). PBIG was applied with a computation time limit of $|V|/5$ seconds to each problem instance. We provide the average results in the first column and the average computation times at which these results were found in the second column. CPLEX was applied with two different computation time limits. In the columns with heading CPLEX we present the results that were obtained with the same computation time limit as PBIG, while the columns with heading CPLEX-L contain the results were the computation time limit was set to 3600 seconds per application. In both cases, the first one of the two columns presents the average of the objective function values of the best solutions found within the computation time limit for

the 10 problem instances of each row. The second column indicates the average optimality gaps (in percent). Note that when the average optimality gap is zero, all 10 corresponding instances were solved to optimality. Finally note that ILP-HEURISTIC was applied with a graph reduction of $X = 20\%$ and with the same computation time limit as CPLEX-L. The best result of each table row is shown with gray background.

The experimental results allow us to make the following observations:

- Concerning the comparison between GREEDY1 and GREEDY2 it can be observed that, generally, GREEDY2 outperforms GREEDY1 in the context of neutral graphs and edge-oriented graphs. Only in the context of node-oriented random graphs GREEDY1 outperforms GREEDY2. This shows that, generally, it is a good idea to take the edge weights already into account during the construction of a solution. Only when the edge weights are not important in comparison with the node weights—that is, in the context of node-oriented graphs—GREEDY1 has advantages.

- CPLEX is able to obtain very good results for the smallest instances with 100 nodes. However, for larger problem instances, it is not competitive anymore. Increasing the computation time limit to 3600 seconds (CPLEX-L) helps for some of the medium size problem instances, where the results in comparison to CPLEX improve considerably. However, when large problem instances are concerned, CPLEX-L is still not competitive.

- ILP-HEURISTIC improves in all but two cases over GREEDY2. However, this is at the cost of a huge increase in computation time. Moreover, it improves in most cases (especially for what concerns medium and large size instances) over CPLEX and CPLEX-L.

- PBIG is, overall, clearly the best-performing algorithm. It outperforms both greedy heuristics in all cases. Moreover, it outperforms both CPLEX variants and ILP-HEURISTIC for all problem instances with more than 100 nodes. Moreover, in those cases where PBIG is worse than CPLEX, it is only slightly worse. This is with the exception of two cases (node-oriented graphs on 100 nodes with $ep \in \{0.05, 0.15\}$) where the difference is more pronounced.

- Concerning the computation time requirements, the two greedy variants are clearly the fastest methods. However, even PBIG produces its best solutions in a very short computation time.

Summarizing, we can state that the algorithm of choice for small problem instances, no matter the graph density, is CPLEX, whereas for larger problem instances PBIG is clearly the best-performing approach.

7 Conclusions and Future Work

This paper has dealt with an NP-hard problem in graphs, the so-called weighted independent domination problem. We proposed the first integer linear programming model for this problem, together with a heuristic that makes use of this model. Additionally, we presented two different greedy heuristics, and a population-based iterated greedy algorithm which takes profit from the better one of the two greedy heuristics. The results have shown that small problem instances are best solved by applying a general-purpose integer linear programming

Table 2: Computational results.

$\|V\|$	Weight scheme	ep	Greedy1		Greedy2		Cplex		Cplex-L		Ilp-Heuristic		Pbig	
			mean	**time**	**mean**	**time**	**mean**	**gap**	**mean**	**gap**	**mean**	**time**	**mean**	**time**
100	neutral	0.05	3589.1	0.0	3519.1	0.0	3060.4	3.3	3049.8	0.0	3055.6	256.4	3051.0	1.3
		0.15	3014.4	0.0	2981.3	0.0	2470.1	34.4	2354.8	11.6	2343.4	2913.7	2093.0	2.7
		0.25	2883.5	0.0	2796.1	0.0	2195.2	41.1	2070.3	2.9	2069.3	1841.8	2070.9	0.1
	node-oriented	0.05	10465.6	0.0	11756.6	0.0	7715.4	0.0	7715.4	0.0	7715.4	0.5	7888.9	3.5
		0.15	4891.6	0.0	5845.4	0.0	3046.6	0.0	3046.6	0.0	3046.6	4.7	3155.8	1.4
		0.25	3297.5	0.0	3488.9	0.0	1808.4	0.0	1808.4	0.0	1808.4	7.8	1832.4	0.2
	edge-oriented	0.05	25698.7	0.0	22269.3	0.0	14423.9	3.1	14378.7	0.0	14435.2	102.1	14393.8	0.4
		0.15	27528.4	0.0	23404.5	0.0	16161.0	53.5	14634.7	12.9	14665.1	2813.2	14609.0	0.1
		0.25	25451.4	0.0	21770.0	0.0	16633.1	69.8	14841.2	27.8	14405.6	1913.2	14572.9	0.1
500	neutral	0.05	14143.1	0.0	13535.1	0.0	12955.5	56.2	11857.9	51.0	11780.8	3600.0	10154.6	73.9
		0.15	12268.5	0.0	11558.0	0.0	11734.5	73.8	10050.1	69.2	9734.1	3600.0	8283.0	11.4
		0.25	11630.3	0.1	10429.5	0.1	11462.4	79.0	10420.0	76.0	9287.1	3600.0	7689.6	14.6
	node-oriented	0.05	15501.5	0.0	18298.1	0.0	12567.6	53.4	12027.7	50.1	12547.5	3600.0	10264.8	71.6
		0.15	6496.3	0.1	7300.1	0.0	12863.7	82.8	5172.5	56.5	5152.6	3600.0	3718.3	20.3
		0.25	4212.4	0.1	4463.7	0.0	15011.7	86.5	3644.9	53.0	3447.8	3600.0	2667.1	7.9
	edge-oriented	0.05	125357.6	0.0	108178.0	0.0	100020.6	83.7	96350.0	82.4	94027.5	3600.0	69245.0	79.3
		0.15	114951.0	0.1	102365.1	0.0	107834.7	93.9	99377.3	92.4	90399.2	3600.0	64723.1	11.9
		0.25	111012.3	0.1	99018.2	0.0	599267.8	97.8	100750.3	94.6	84572.1	3600.0	64694.0	6.2
1000	neutral	0.05	25569.6	0.0	23489.7	0.0	25156.1	69.9	25156.1	69.4	21547.2	3600.0	17927.0	87.4
		0.15	20827.1	0.2	20689.1	0.2	129091.1	90.1	21117.8	81.0	19772.8	3600.0	15051.7	35.0
		0.25	20858.8	0.3	19280.5	0.3	229070.6	97.5	45126.9	86.5	18553.8	3600.0	14251.3	18.1
	node-oriented	0.05	18048.6	0.1	20142.3	0.1	35766.3	83.4	28123.1	76.4	16566.6	3600.0	11434.6	167.1
		0.15	7408.3	0.2	7987.4	0.2	40581.6	95.3	40581.6	93.2	7987.4	3600.0	4617.8	88.8
		0.25	4941.9	0.4	5566.6	0.3	35838.9	100.0	33023.7	93.4	5566.6	3600.0	3535.8	53.1
	edge-oriented	0.05	238600.0	0.1	202992.0	0.1	215587.9	91.9	209391.8	91.0	198508.6	3600.0	132097.2	149.5
		0.15	209709.3	0.2	182726.6	0.4	1242486.5	98.6	204521.2	96.7	179139.9	3600.0	127561.7	49.7
		0.25	198537.0	0.4	181150.0	0.3	1581837.5	99.0	903076.4	98.3	178003.7	3600.0	126145.5	21.3

solver. Medium and large scale instances, on the other side, are best solved by the population-based iterated greedy approach.

In the near future we plan to investigate if there are better ways to take profit from the developed ILP model in a heuristic way, for example, in the context of a large neighborhood search algorithm or another hybrid algorithm called construct, merge, solve and adapt.

References

[1] 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.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] G. Lin, W. Zhu, and M. M. Ali. An effective hybrid memetic algorithm for the minimum weight dominating set problem. *IEEE Transactions on Evolutionary Computation*, 2016. In press.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.