Available online at www.sciencedirect.com
ScienceDirect
Mathematics and Computers in Simulation 207 (2023) 41–58
MATHEMATICS AND COMPUTERS IN SIMULATION
www.elsevier.com/locate/matcom

Original articles

An iterated greedy algorithm for finding the minimum dominating set in graphs

A. Casado[a], S. Bermudo[b], A.D. López-Sánchez[b], J. Sánchez-Oro[a,*]

[a] *Computer Science Department. Universidad Rey Juan Carlos, Tulipán s/n. 28933 Móstoles, Madrid, Spain*
[b] *Department of Economics, Quantitative Methods, and Economic History. Universidad Pablo de Olavide, Ctra. de Utrera km 1. ES-41013 Sevilla, Spain*

Received 22 July 2022; received in revised form 29 October 2022; accepted 23 December 2022
Available online 29 December 2022

Abstract

A dominating set in a graph is a set of vertices such that every vertex outside the set is adjacent to a vertex in the set. The domination number is the minimum cardinality of a dominating set in the graph. The problem of finding the minimum dominating set is a combinatorial optimization problem that has been proved to be $\mathcal{NP}$-hard. Given the difficulty of this problem, an Iterated Greedy algorithm is proposed for its solution and it is compared to the solution given by an exact algorithm and by the state-of-art algorithms. Computational results show that the proposal is able to find optimal or near-optimal solutions within a short computational time. Specifically, from the set of instances which can be optimally solved, the proposed method presents an average deviation of 0.04%. Regarding the more complex set of instances, where the exact method is not able to reach the optimal value, the proposed method achieves an average deviation of 1.23% with respect to the best-known solution.

Keywords: Domination number; Minimum dominating set; Greedy heuristics; Iterated greedy; Exact algorithm

1. Introduction

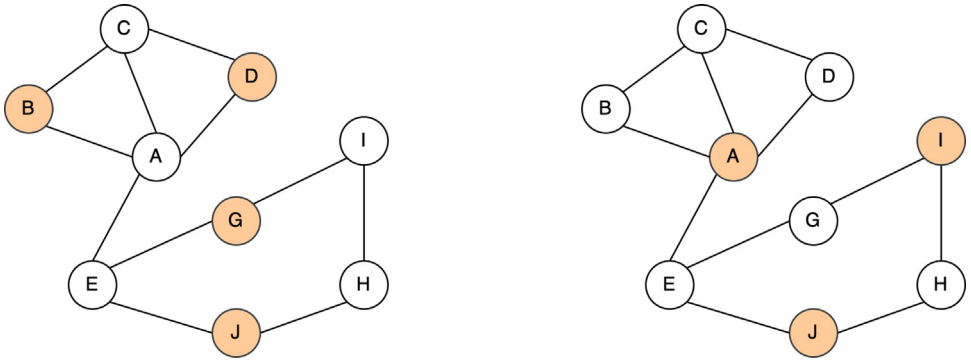
Within the last sixty years and due to the development of large networks such as road networks, social networks, electrical networks, communication networks, computer networks or security networks among others, the graph theory has been the focus of interest for many researchers and practitioners. In graph theory, the study of domination and related subset problems, such as matching, independence or covering has had a significant growth. In particular, the problem of finding dominating sets in graphs started in 1960 (see [7]), but it was in 1962 when the concept of domination number of a graph was defined in [20]. Since then, more than 2000 research papers have been published on this topic. We refer the reader to [13,14,30] for comprehensive surveys.

1.1. Problem definition

A dominating set in a graph is a set of vertices such that every vertex outside this set is adjacent to at least a vertex in the set. The application of domination in graph lies in various fields for solving real life problems. It

* Corresponding author.
E-mail address: jesus.sanchezoro@urjc.es (J. Sánchez-Oro).

https://doi.org/10.1016/j.matcom.2022.12.018



(a) A minimal dominating set D_1. (b) A minimum dominating set D_2.

Fig. 1. Example of a minimal and a minimum dominating sets.

includes social networks (see [29]), radio stations (see [9]), computer communication networks (see [31]), network surveillance (see [12]), sets of representatives, etc. Fig. 1 shows an example with 9 vertices. Notice that the set $D_1 = \{B, D, G, J\}$, depicted in Fig. 1(a), is a minimal dominating set (it does not contain a proper subset which is a dominating set), while the set $D_2 = \{A, I, J\}$, depicted in Fig. 1(b), is a minimum dominating set or a γ-set (a dominating set with the smallest cardinality) for the considered graph.

Many real-world situations fits into domination in graphs. For instance, let us consider a graph whose vertices represent a group of people and edges between two people means that they know each other. The problem of determining a good set of representatives consists on finding the smallest group of people such that everyone not on it has a known person in that group. Therefore, the solution is a dominating set with the smallest cardinality (see [13]).

A similar application could be used to monitor a computer network or to send information through a computer network. In this example, the computer network is modeled by a graph where vertices represent computers and edges are direct communication links between pairs of computers to pass information. The problem considers that from time to time it is needed to collect information from all computers. Therefore, it is needed to identify a small set of computers, usually known as backbone, connected to all other computers to collect all the information (see [13]).

Nowadays, the most popular application of domination in graphs lies in a graph representing a social network, such as Facebook, Twitter or Instagram, since they serve as an important medium for communication and information disseminating. Here, vertices represent users of the social network and edges are friendship between users on the social network. The popularity of social networks makes them a very powerful tool in the viral marketing of products and political campaigns. Many authors proposed different influence maximization problems [6,8,16] where a particular case is the problem of determining the smallest group of users able to influence the rest (thanks to the friendship connections). Therefore, the objective of finding dominating sets is intimately related with the influence maximization problem because a dominating set is a direct way to reach every user in a social network, so it is a cascade model with only one step (see [6,8,16]).

More formally, let $G = (V, E)$ be a simple graph of *order* $n = |V|$ and *size* $m = |E|$, being $|V|$ and $|E|$ the cardinality of the vertex and edge sets, respectively. A set $S \subseteq V$ is a *dominating set* in G if every vertex in $V \setminus S$ is adjacent to a vertex in S. A set $D \subseteq V$ is a *minimal dominating set* in G if it is a dominating set and it does not contain a proper subset which is a dominating set. The minimum cardinality among all dominating sets is called the *domination number*, denoted by $\gamma(G)$, and a dominating set with that cardinality is called a *minimum dominating set* or a *γ-set*. The problem of finding the minimum dominating set in a graph is known as the Minimum Dominating Set Problem (hereinafter MDSP).

There are a finite number of possible dominating sets of G, as maximum $2^n - 1$ sets. To determine the value of $\gamma(G)$ and obtain the minimum dominating set $D^\star$, it is sufficient to enumerate all subsets of V in non-decreasing order of cardinality and check, for any given subset $S \subseteq V$, whether S is a dominating set. Therefore, the value of $\gamma(G)$ is the cardinality of the first dominating set found. Although such enumeration algorithm is easy, unfortunately, it requires $O(2^n)$ steps in the worst case, that is, it has an exponential time complexity that depends on the order

of the graph. To date, as far as we know, the best exact algorithm able to find the minimum dominating set for arbitrary graphs was proposed by Fomin et al. (see [10]) and it has a complexity $O(1.5137^n)$.

1.2. Mathematical model

Next, a mathematical model based on an Integer Linear Programming (ILP) formulation is included in order to formalize the MDSP. Let $A = (a_{ij})$ be the adjacency matrix of the considered graph G. As it is well-known, the adjacency matrix is a square $n \times n$ matrix such that the element a_{ij} is equal to one when there is an edge from vertex i to vertex j, and zero otherwise. Therefore, this model uses binary variables x_i that take value 1 if vertex i is selected and 0 otherwise. The problem can be formulated as follows:

$$\min \sum_{i=1}^{n} x_i \tag{1}$$

$$\text{s.t.} \sum_{i=1}^{n} a_{ij} x_i \geq 1 \quad \forall j = 1, \ldots, n, \tag{2}$$

$$x_i \in \{0, 1\} \quad \forall i = 1, \ldots, n. \tag{3}$$

The objective function (1) represents the total number of selected vertices while constraints (2) ensure that each non-selected vertex in G has a neighbor which is selected. Finally, constraints (3) define the nature of the decision variables.

Since the problem of finding the minimum dominating set is $\mathcal{NP}$-hard for arbitrary graphs (see, for instance, [14]), the use of metaheuristics to find such γ-sets is more than justified. Although there are only a few works focused on finding an algorithm to approximate the domination number (see [4,15,22,23,28]), the problem has been extensively studied for particular graphs, and many bounds for this parameter have been given for general graphs (see, for instance, [13,14]).

1.3. Metaheuristics for the MDSP

Metaheuristics are one of the most extended approximate algorithms for solving hard and complex combinatorial optimization problems and they can be defined as strategies that guide the search process to escape from local optima. These are robust algorithms able to provide high-quality solutions in reasonable computing times without guaranteeing optimality. Its use is justified when it is not possible to compute the optimal solution with an exact algorithm.

Although there exist several taxonomies, the most extended one classifies metaheuristics in two different categories: trajectory-based and population-based metaheuristics. The former maintains a single solution during the search, performing different movements from one solution to another one in the search space in an iterative way, with the aim of attaining a good solution along the trajectory; while the latter maintains a population of solutions in the search, using the information of the entire population to guide the search.

This work is focused on designing a metaheuristic algorithm able to approximate the minimum dominating set and, therefore, the domination number for arbitrary graphs, even for graph mining, that are large-scale real-world graphs. Specifically, an Iterated Greedy (IG) algorithm is proposed to address this problem based on the idea of starting from an initial solution that will be destructed and reconstructed in an iterative way until a stopping criterion is met (more details are included in Section 2).

There are several works in the literature addressing the minimum dominating set from a metaheuristic perspective. The most recent paper [4] proposes an order-based Randomized Local Search (RLS) algorithm, that represents a solution based on permutations of vertices which are transformed into dominating sets by means of a greedy algorithm. Then, if S is a solution and we consider the vertices of S as the first $|S|$ vertices in V, this solution is improved using a simple movement, named jump perturbation operator, that selects a position $j \in \{2, 3, \ldots, n\}$, put v_j in the first position and shift one position to the right all vertices between positions 1 and $j - 1$. The author solves the problem for unit disk graphs, scale-free networks, real-world graphs including samples of two social networks and graphs studied in the field of network science, and he performs a comparison to the state-of-the-art algorithms.

Next, the state-of-the-art algorithms are briefly mentioned. The first one is an Ant Colony Optimization Algorithm hybridized with a Local Search (ACO-LS) proposed by [22] that randomly generates a population of solutions that evolves following a probabilistic scheme known as the pheromone value of ants associated with a vertex. The LS consists on removing redundant vertices of a solution. Then, an extension of the Ant Colony Optimization algorithm hybridized with a Local Search with Pre-Processing (ACO-PP-LS) is proposed by [23], that generates a population of solutions using a greedy algorithm obtaining solutions that are not totally random, thus accelerating the convergence of ACO-LS algorithm. Furthermore, a new variant of ACO-LS, named ACO-LS-S, is presented [4], which selects the vertices involved in each transition and initialize the pheromones following a greedy criterion.

The main contributions of this paper are detailed next:

- We propose an algorithm able to solve the minimum dominating set problem for any graph without a specific structure.
- The algorithm leverages the topology of the graph identifying support and leaf vertices with the aim of reducing the solution space, thus accelerating the algorithm.
- Two constructive procedures are proposed: the first one iteratively generates a solution from scratch, while the second one follows the opposite approach, i.e., it starts from a complete graph and iteratively removes the least promising vertices.
- A post-processing method is presented to remove redundant vertices from the explored solutions, which accelerates the algorithm.
- An efficient local search is presented, which is able to perform only the most promising moves by predicting if a move will end in a feasible solution.
- We have significantly enlarged the testbed of instances in order to make more robust comparisons.

This work is organized as follows. Section 2 describes the algorithm implemented to solve the problem under consideration. Section 3 presents the computational results performed to test the quality of the proposal, including a comparison against exact algorithms. Finally, Section 4 summarizes the paper and discusses future works.

2. Algorithmic approach

This Section is devoted to provide a thorough description of the proposed algorithm designed to address the MDSP. Specifically, a metaheuristic approach based on IG framework with the aim of providing high-quality solutions in short computing times. We have opted by a metaheuristic since the MDSP is $\mathcal{NP}$-hard and an exact procedure is not able to deal with complex instances. Therefore, the metaheuristic approach emerges as the most adequate method for solving large and complex instances in the context of MDSP.

IG is a metaheuristic framework originally proposed for solving a scheduling problem [25] and it has been in continuous evolution, proving its efficiency for solving a wide variety of hard combinatorial optimization problems. For instance, IG has been considered for solving community detection problems [27], robotic flowshop scheduling problems [18], or regenerator location problems [24], among others. The success of IG relies in the simple yet effective idea of iteratively destructing and then reconstructing the incumbent solution to escape from local optima. Algorithm 1 depicts the pseudocode of IG framework.

The algorithm receives three input parameters, namely: $G = (V, E)$, the input graph; β, the percentage of vertices removed in the destruction phase; and Δ, the maximum number of iterations without improvement that IG allows to perform (stopping criterion).

IG starts by creating an initial solution using one of the constructive procedures proposed in Section 2.1 (step 1). Note that this initial solution is a minimal dominating set. Then, a local optimum is found, starting from this initial solution, by applying the local improvement method presented in Section 2.2 (step 2), becoming D_b the best solution found so far, in our case, resulting in an eventually better minimal dominating set. The method then iterates until reaching a fixed stopping criterion (steps 4–14). In the context of MDSP, the stopping criterion considered is a maximum number of iterations performed without finding any improvement, which is controlled by the input parameter Δ. In each iteration, the incumbent solution D_b is partially destroyed with one of the destruction procedures described in Section 2.3, generating a solution D_d (step 5). Notice that D_d is an unfeasible solution since the partial destruction performed results in a set that is not a minimal dominating set anymore and not even a dominating set, as it will be later discussed in Section 2.3. Then, in order to attain a new minimal dominating set, the feasibility is fixed by means of one of the reconstruction procedures presented in Section 2.4, resulting in

Algorithm 1: IG($G = (V, E)$, β, Δ)

1: $D \leftarrow InitialSolution(G)$
2: $D_b \leftarrow LocalImprovement(D)$
3: $\delta \leftarrow 0$
4: **while** $\delta < \Delta$ **do**
5: $D_d \leftarrow Destruction(D_b, \beta)$
6: $D_r \leftarrow Reconstruction(D_d)$
7: $D_i \leftarrow LocalImprovement(D_r)$
8: **if** $|D_i| < |D_b|$ **then**
9: $D_b \leftarrow D_i$
10: $\delta \leftarrow 0$
11: **else**
12: $\delta = \delta + 1$
13: **end if**
14: **end while**
15: **return** D_b

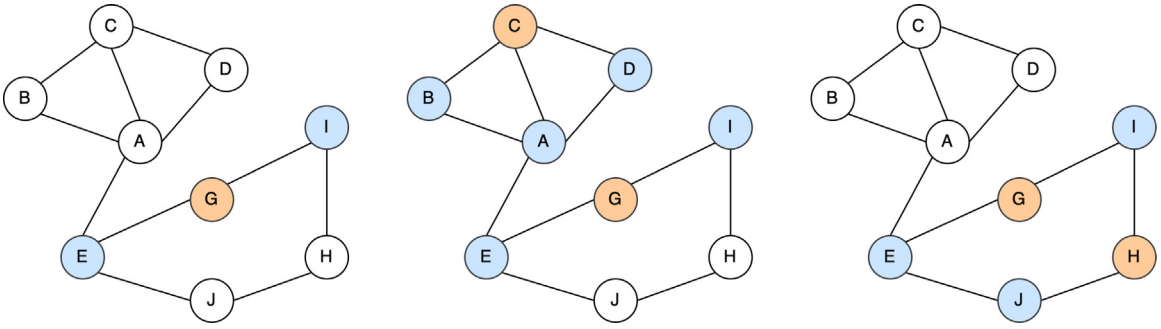
a new feasible solution D_r (step 6). The reconstructed solution D_r is not necessarily a local optimum, so the local improvement method is applied to further improve it, resulting the solution set D_i, which is a minimal dominating set again (step 7). At the end of each iteration, the objective value of the set D_i is compared with the objective value of the best set found so far D_b (step 8). Notice that, as it was aforementioned in Section 1, the objective function value is evaluated as the cardinality of every set. If D_i outperforms D_b, then D_b is updated (step 9), and the number of iterations without improvement is reset (step 10). Otherwise, the number of iterations without improvement is incremented for the next iteration (step 12). The method ends when reaching the maximum number of iterations without improvement, i.e., $\delta = \Delta$, returning as solution the best set found during the search (step 15).

2.1. Initial solution

IG requires from an initial solution to start the search from. Although the literature suggests that even a random solution is enough for IG to provide high-quality solutions, several recent research have shown that starting from a promising region of the search space usually lead to most of metaheuristic algorithms to either reduce the computing time required or to find better solutions [2,21]. Therefore, two different greedy procedures are proposed to generate an initial feasible solution for IG which, in our case, is a minimal dominating set. The first procedure is based on the greedy insertion of vertices at each iteration, meanwhile the second one is based on greedy deletion of vertices at each iteration.

It is important to remark that, leveraging the information derived from the MDSP, there are some specific vertices that can be always taken in optimal solutions for the MDSP, the so-called support vertices. Before defining a support vertex, it is necessary to introduce some preliminary concepts. Let us define $N(v)$ as the set of adjacent vertices to v, and $N[v] = N(v) \cup \{v\}$. For any $D \subseteq V$, we denote $N(D) = \bigcup_{v \in D} N(v)$ and $N[D] = N(D) \cup D$, and we say that a vertex v is dominated by D if $v \in N[D]$. A *leaf vertex* is a vertex v such that $|N(v)| = 1$, and a *support vertex* is a vertex adjacent to a leaf. We denote by L the set of *leaf vertices* of the graph and by SV the set of support vertices.

Therefore, the first procedure follows a traditional greedy approach which starts from an empty solution and it iteratively selects the most promising vertex to be included in the next iteration until reaching a feasible solution. However, instead of starting from an empty solution, it starts from a solution in which all support vertices are already included $D = SV$, and it iteratively adds new vertices until the incumbent solution becomes a dominating set. Analogously, the leaf vertices are never considered to be part of the solution, since they are always dominated by a support vertex. This idea allows the procedure to start from a partial solution, reducing its computational effort. The experiments will show the effect of considering support vertices in the performance of the algorithm. We will refer this methodology as the *Greedy Insertion Procedure* (GIP).



(a) $D = \{\texttt{G}\}$. (b) Evaluation of vertex C. (c) Evaluation of vertex H.

Fig. 2. Evolution of GIP using g_{GIP} to evaluate vertices C and H. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

As it is customary in greedy algorithms, the selection of the next vertex strictly depends on a greedy function which is usually designed specifically for the problem under consideration. In the context of MDSP, the greedy function $g_{GIP}(v)$ selected tries to evaluate the contribution of a vertex v to the solution under construction D. In particular, it is evaluated as the number of adjacent vertices of v which are not dominated by D (since those vertices will be dominated by v if it is finally selected). In mathematical terms,

$$g_{GIP}(v) = |N[v] \setminus N[D]|$$

Fig. 2(a) shows the performance of GIP when starting from $D = \{\texttt{G}\}$, where the selected vertices are colored in orange and the dominated vertices are colored in blue. On the one hand, Fig. 2(b) shows the evaluation of the greedy function when considering vertex C. In particular, selecting vertex C would result in covering vertices B, A, D, and C, which were not previously dominated, resulting in $g_{GIP}(\texttt{C}) = 4$. On the other hand, Fig. 2(c) illustrates the evaluation of greedy function over vertex H. In this case, the new dominated vertices are J and H, since I was already dominated by G, resulting in $g_{GIP}(\texttt{H}) = 2$. In this example, C is selected since $g_{GIP}(\texttt{C}) > g_{GIP}(\texttt{H})$.

Having defined the greedy function, the GIP iteratively selects the vertex with the maximum greedy function value until reaching a feasible solution, i.e., a solution in which all vertices are dominated. Therefore, the process ends when $g_{GIP}(v) = 0$, $\forall v \in V \setminus D$.

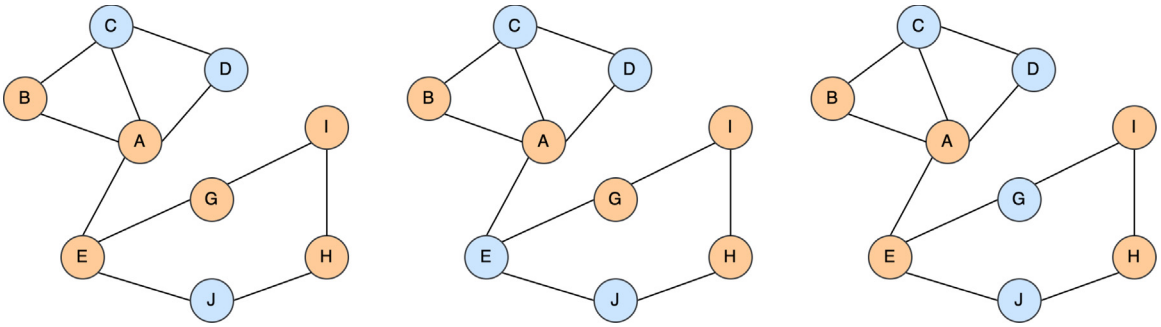
The second procedure proposed follows an opposite approach. Instead of starting from scratch (but for the support vertices), and iteratively selecting a new vertex to be included in the solution, this procedure considers that the solution initially contains all vertices of the graph. Then, the method iteratively removes a vertex from the solution until reaching a state in which the removal of any vertex results in an unfeasible solution. Similarly to the GIP, the support vertices are not considered to be removed, and the leaf vertices are not initially included in the solution. Therefore, the method actually starts with $D = G \setminus L$, where L is the set of leaf vertices. Then, the *Greedy Deletion Procedure* (GDP) iteratively removes the least promising vertex to be kept in the solution by means of a different greedy function value.

In this case, it is interesting to remove the vertex v such that the vertices in its closed neighborhood $N[v]$ remain dominated by as many vertices as possible. The rationale behind this is that this criterion will eventually allow more vertices to be removed in future iterations. More formally,

$$g_{GDP}(v) = \min_{u \in N[v]} |(D \setminus \{v\}) \cap N[u]|$$

Following this greedy function, the GDP iteratively selects the vertex v with the largest value of $g_{GDP}(v)$, until no vertex can be removed without guaranteeing the feasibility of the solution.

Fig. 3(a) shows the behavior of GDP when starting from solution $D = \{\texttt{A}, \texttt{B}, \texttt{E}, \texttt{G}, \texttt{H}, \texttt{I}\}$, following the same coloring as in Fig. 2. In Fig. 3(a) the greedy function g_{GDP} is evaluated over vertex E. The adjacent vertices to E are A, G, and J. If E is removed, the vertex A is still dominated by vertices A and B; the vertex G is dominated by G and I; the vertex E is dominated by G and A; and the vertex J is dominated by H. Therefore, the value of $g_{GDP}(\texttt{E}) = 1$, since it is the minimum number of vertices that cover a vertex of $N[\texttt{E}]$. Fig. 3(c) shows a similar



(a) $D = \{\texttt{A,B,E,G,H,I}\}$. (b) Evaluation of vertex E. (c) Evaluation of vertex G.

Fig. 3. Evolution of GDP using g_{GDP} to evaluate E and G.

evaluation over vertex G, whose adjacent vertices are E and I. In the case of E, it remains dominated by A and E, vertex G is dominated by E and I, while vertex I is dominated by H and I. Then, the greedy function value is $g_{GDP}(\texttt{G}) = 2$. Therefore, the vertex G will be selected to be removed by this procedure before selecting vertex E.

Up to this point, a solution generated either with GIP or with GDP is always a dominating set. However, in the case of GIP, it is not guaranteed that it is a minimal dominating set. In order to assure that any solution derived from GIP or GDP is a minimal dominating set, we propose a procedure that checks if some of the vertices belonging to the dominating set are redundant. To be more accurate, the procedure checks if a vertex $v \in D$ and all their adjacent vertices are dominated by $D \setminus \{v\}$ and, if so, v will be removed from the dominated set. With the aim of checking and removing those redundant vertices, the *checking procedure* (CheckP) traverses all vertices in the dominating set D and removes all redundant vertices. A vertex $v \in D$ is redundant if and only if $|(D \setminus \{v\}) \cap N[u]| \geq 1, \forall u \in N[v]$. In this case, v can be definitively eliminated from D as long as the above condition is met.

2.2. Local improvement

In IG framework, both the initial solution and the one resulting in each iteration after the reconstruction phase are not necessarily local optima. Therefore, a local improvement method is proposed to further improve those solutions which may eventually lead to better results.

The improvement phase can be performed by considering complex methods such as Tabu Search [2], Variable Neighborhood Search [19] or any other metaheuristic. However, these strategies usually result in rather computationally demanding algorithms. In order to maintain an efficient algorithm, a simple yet effective local search procedure is proposed.

The definition of a local search is given by its three main components: the move operator considered, the neighborhood explored, and the strategy followed to explore the neighborhood. In the context of MDSP, the move operator is defined as an exchange move. This movement consists of removing a vertex from the solution under evaluation, replacing it with a new vertex which is not already included in the solution. More formally,

$$Exchange(D, u, v) \leftarrow (D \setminus \{u\}) \cup \{v\}$$

This move operator is only considered if the resulting solution is feasible. If the removal of vertex u and the insertion of vertex v leaves one or more vertices without being dominated, then the move is undone to maintain the original feasible solution.

Notice that, since the evaluation of the objective function is the cardinality of the solution, the exchange move will never lead to an improvement by itself. However, after performing an exchange move, some of the previously selected vertices may be redundant, therefore the checking procedure is again applied to remove them and improve the value of the objective function.

Hence, an improvement move in the context of MDSP is defined as an exchange move whose corresponding checking procedure is able to remove, at least, one vertex.

Having defined the move operator, the neighborhood of a solution can be directly derived from it. In particular, the local improvement phase considers the neighborhood $N_e(D)$, which is conformed with all feasible solutions that can be reached by performing an exchange move. In mathematical terms,

$$N_e(D) \leftarrow \{D' = Exchange(D, u, v), \forall u \in D \wedge \forall v \in V \setminus D\}$$

Once the neighborhood has been defined, it is necessary to establish the strategy used to traverse the neighborhood. There are two traditional strategies that can be found in the literature: best improvement and first improvement. The former traverses the neighborhood performing, in each step, the move that leads to the best solution in the neighborhood. The latter, on the contrary, performs the first move that leads to an improving solution in the neighborhood. Both of them stop when no improvement is found.

In this research first improvement is considered since it has been experimentally shown that it is able to reach similar results than best improvement but being less computationally demanding [3,11]. In the context of first improvement, the order in which the neighborhood is traversed may affect the results. In order to diversify the search, we consider a random order for traversing the neighborhood.

Analyzing the described Local Search (LS), it can be seen that several moves considered in the search lead to an unfeasible solution. Therefore, it would be desirable to perform only those moves that result in a feasible solution. To that end, an efficient version of the local search procedure is proposed. This Efficient Local Search (ELS) is able to identify which moves lead to a feasible solution before actually performing the move.

This optimization of the local search procedure consists of, given the vertex to be removed from the solution, identify which are the candidate vertices that can replace the one that will be removed. Let us consider that $u \in D$ and $v \notin D$. To remove vertex u from the solution and replace it by vertex v, the necessary condition is that every vertex dominated only by u, i.e., every vertex $w \in N[u]$ such that $|D \cap N[w]| = 1$, is dominated by v, i.e., $w \in N[v]$.

Given this definition, the efficient local search only considers those moves that lead to feasible solutions, thus reducing the number of operations performed without deteriorating the solution quality. The experimental section will show the effect of this optimization in the final algorithm.

2.3. Destruction phase

The destruction phase of IG algorithm is one of the responsible stages for diversifying the search by destroying a certain part of the solution. Destroying (partially) a solution in the context of MDSP basically consists of removing some of the selected vertices. The number of vertices to be removed depends on the input parameter β. With the aim of having a scalable algorithm, the value of β is a percentage of the size of the solution (or $\beta \in [0, 1]$), and it will be discussed in Section 3.

The destruction phase can be performed following either a random or greedy criterion. The former focuses on diversification and leads the algorithm to different regions of the search space while the latter, focused on intensification, tries to find a better solution in the region under exploration. In this research, both alternatives are presented to evaluate the effect of each one of them in the final algorithm.

The first variant, the *random destruction*, removes β of the vertices from the incumbent solution (or $\beta \cdot |D|$ vertices from the incumbent solution). Those vertices are selected at random, resulting in an unfeasible solution, which will become feasible after applying the reconstruction phase presented in Section 2.4. This random selection allows the algorithm to consider solutions that, otherwise, would be never explored.

The second variant, named *greedy destruction*, also removes β of the vertices from the solution (or $\beta \cdot |D|$ vertices from the solution), but those vertices are selected following a greedy criterion. The rationale behind this is that selecting the vertices whose contribution to the solution is minimal will eventually reduce the size of the dominating set. The greedy criterion followed in this phase is similar the one used in the GDP presented in Section 2.1. In particular, the method iteratively removes the vertex v whose adjacent vertices (which are those dominated by v) are dominated by the largest number of vertices belonging to $D \setminus \{v\}$.

2.4. Reconstruction phase

The reconstruction phase is designed to recover a feasible solution after having destructed the incumbent one in the destruction phase described in Section 2.3. In order to do so, two different approaches can be followed.

On the one hand, if the reconstruction phase is devoted to diversify, it can be performed by randomly adding vertices to the incumbent solution until all vertices are dominated, i.e., a feasible solution is reached. On the other hand, a greedy criterion can be followed to minimize the number of required vertices to be included in the solution, thus reducing the objective function value.

Therefore, it is necessary to establish a greedy criterion to select the next vertex to be included in the solution being reconstructed. Analogously to the destruction phase, the greedy criterion selected is the same as the one considered in the GIP presented in Section 2.1. Specifically, the vertex selected is the one with the largest number of adjacent vertices which are not yet dominated. This criterion maximizes, in each step, the number of new vertices dominated, thus reducing the number of required dominating vertices.

It is expected that the *random reconstruction* results in more diverse solutions but with a larger number of dominating vertices, while the *greedy reconstruction* generates less diverse solutions but with a reduced number of dominating vertices. The best reconstruction method is tested in Section 3.

2.5. Structural complexity

This Section is devoted to describe the structural complexity of each part of the proposed algorithm and in turn, the complexity of the complete algorithmic proposal. Although the *Big O* notation is used, it is worth mentioning that a good design and implementation of the algorithm usually leads to reduced computing times, as it can be seen in Section 3.

Without loss of generality, let us define p as the cardinality of a given solution. Since the support and leaf vertices are not considered in any part of the procedure, we denote by $\hat{n}$ the number of vertices in the graph which are not support or leaf vertices, and by $\hat{p}$ the number of selected vertices in the solution which are not support vertices. Notice that this value is considerably smaller than n (see Table 2 for an example on the solution size reduction using support vertices).

First of all, the constructive procedures will be analyzed. In this case, both constructive procedures present the same structural complexity. Then, the GIP iteratively adds a new vertex until the solution becomes feasible, with a complexity of $O(\hat{p})$. In each iteration, the next best vertex is selected, with a complexity of $O((\hat{n}-\hat{p})\cdot\hat{n})$, since all non-selected vertices $\hat{n}-\hat{p}$ are traversed and then their adjacent vertices, which could eventually be $\hat{n}$. Then, the complete constructive procedure has a complexity of $O(\hat{p}\cdot(\hat{n}-\hat{p})\cdot\hat{n})$. The same analysis is performed for the GDP.

The next method to be analyzed is the `CheckP`, which traverses the set of selected vertices in the solution, $O(\hat{p})$ and, for each one of them, the adjacent vertices are checked $O(\hat{n})$, resulting in a complexity of $O(\hat{p}\cdot\hat{n})$.

The local search method is the most complex procedure in any metaheuristic algorithm since it requires to traverse a complete neighborhood in several occasions. In this section, only the efficient local search complexity is analyzed since it is the main contribution of this research in the context of local search methods. Each iteration of the efficient local search traverses the set of selected vertices, $O(\hat{p})$, and, for each vertex its neighbors are analyzed, $O(\hat{n})$. Additionally, if the move can be performed (which occurs in a reduced number of iterations), the checking procedure is applied, $O(\hat{p}\cdot\hat{n})$. Then, the complexity of an iteration of the local search procedure, if an improvement is found, is $O(\hat{p}\cdot\hat{p}\cdot\hat{n})$, while an iteration where no improvement is found results in a complexity of $O(\hat{p}\cdot\hat{n})$.

The destruction phase of IG requires to remove a certain number of selected vertices, which is $\beta\cdot\hat{p}$, resulting in a complexity of $O(\beta\cdot\hat{p})$. Notice that $\beta\cdot\hat{p}\ll\hat{p}$. Conversely, the reconstruction phase iteratively adds a new vertex to the incumbent solution until it becomes feasible, requiring, in the worst case, $\hat{n}-\hat{p}$ iterations. Notice that this case indicates that all vertices have been included, which is an extremely rare situation, as it can be seen in the results. Therefore, the complexity of the reconstruction phase is $O(\hat{n}-\hat{p})$.

Summarizing, a single iteration of our IG algorithm performs a destruction phase, $O(\beta\cdot\hat{p})$; a reconstruction phase, $O(\hat{n}-\hat{p})$; a checking, $O(\hat{p}\cdot\hat{n})$; and a local search method, $O(\hat{p}\cdot\hat{p}\cdot\hat{n})$. Then, the complexity of an IG iteration is evaluated as the maximum complexity of the inner methods, which is the local search complexity, $O(\hat{p}\cdot\hat{p}\cdot\hat{n})$. Notice that IG is executed a maximum number of iterations without improvement, which cannot be analyzed from a complexity point of view.

Table 1
Comparison of proposed constructive procedures.

Algorithm	Avg.	Time	Dev. (%)	#Best
GIP	**751.67**	**0.13**	**0.01**	**8**
GDP	1070.00	5.76	0.22	2
RND	2623.33	0.37	1.90	0

3. Computational results

This Section presents and discusses the results of the computational testing conducted with the algorithm proposed in this paper, IG algorithm. A total of 76 instances are solved on an AMD EPYC 7282 (2.8 GHz) with 16 GB RAM, and the heuristic algorithms were implemented using Java 11, while the exact approach has been implemented using Gurobi 9. The set of instances is divided into two different subsets: 28 instances are directly derived from the literature to perform a fair comparison, while new and more challenging 96 instances are generated in order to evaluate the limits of both the exact and metaheuristic approaches. The best method found in the literature is the order-based Randomized Local Search (RLS) algorithm [4], which considers the set of 28 instances aforementioned, divided in: unit disk graphs, random scale-free networks generated by Barabási–Albert model [1], and a set of real-world graphs, including samples from two social network services, graphs studied in network science, as well as several DIMACS graphs. All instances and source code are publicly available at https://grafo.etsii.urjc.es/MDSP.

The experiments are divided into two different phases: preliminary and final experiments. The former is devoted to tune the input parameters of the algorithm and the best strategy to conduct the search, while the latter is designed to perform a direct comparison with the state-of-the-art algorithm, with the aim of evaluating the contribution of our proposal. In order to avoid overfitting, the preliminary experimentation is performed over a subset of 9 representative instances derived from the original set of 28 instances.

All tables report the following metrics: Avg., the average objective function value obtained by each algorithm over the set of considered instances; Time, the average computing time required by the algorithm (in seconds); Dev., the average percentage deviation with respect to the best solution found in the experiments; and #Best, the number of times that each algorithm reaches the best solution of the experiment. Additionally, the best result of each experiment is highlighted in bold font.

3.1. Preliminary experimentation

The first preliminary experiment is devoted to select the best procedure to get the initial solution, in our problem, a minimal dominating set as already explained in Section 2.1. To that end, two methods are considered: the GIP and the GDP. Additionally, the comparison includes a third procedure, denoted as RND, that generates a feasible solution by selecting a vertex at random in each iteration. This random method is included just for illustrative purposes, to show the relevance of designing an specific algorithm for the MDSP. Table 1 shows the results obtained by each constructive procedure.

As it can be derived from Table 1, both the GIP and the GDP obtain consistently better results than the RND. It is worth mentioning that the results obtained by the random procedure also consider that the support vertices are in the solution and the leaf vertices are not, thus starting from a partially constructed solution. Comparing the GIP and the GDP, the differences in computing time are remarkable (0.13 versus 5.76 s on average). This is partially explained because, in the MDSP, the number of vertices required to have a feasible solution is small with respect to the order of the graph, so, generally it will require from more iterations. Additionally, the constructive procedure is able to reach 8 out of 9 best solutions with an average deviation of 0.01%, indicating that in the solution in which it is not able to reach the best solution, it still remains really close to it. This behavior, together with the reduced computing time required, lead us to select the constructive procedure as the best method to generate an initial solution.

The aim of the following experiment is to evaluate the effect of the efficient local search and the naïve local search. Specifically, this comparison is done with respect to the computational effort required to find a local optimum when starting from the same initial solution (the one generated by the GIP). As expected, both methods will obtain

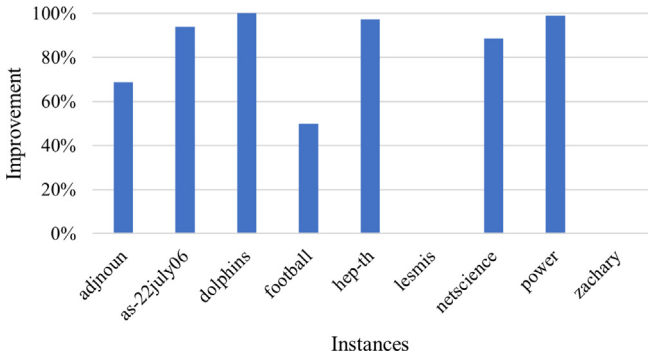


Fig. 4. Percentage of improvement in time obtained when using the efficient local search method.

Table 2
Percentage of support vertices in each instance and percentage of dominated vertices if support vertices are included in the solution.

Instance	$\frac{\lvert SV\rvert}{\lvert V\rvert}\%$	$\frac{N[SV]}{\lvert V\rvert}\%$
`adjnoun`	50.00	80.36
`as-22july06`	91.56	98.95
`dolphins`	46.67	69.35
`football`	0.00	0.00
`hep-th`	74.57	72.11
`lesmis`	70.00	80.52
`netscience`	65.41	51.23
`power`	62.11	68.97
`zachary`	25.00	50.00
Average	53.92	63.50

the same results in terms of quality, however, it would be interesting to compare the computing time required to find a local optimum. To graphically visualize this performance, Fig. 4 shows, for each instance, the percentage of improvement in computing time obtained by ELS when comparing it with the naïve LS method.

Results show the relevance of considering the efficient local search, being, on average, 66% faster than the naïve local search. Moreover, except from instances `lesmis` and `zachary`, in which no improvement in terms of computing time is found, the improvement in computing time is 85% on average. It is worth mentioning that those two instances are the smallest ones, being the most easily solved by the local search method. Therefore, the contribution of the efficient local search is confirmed.

Once that the procedure to generate the initial solution and the efficient local search improvement have been configured, the next experiment tries to analyze the effect of including the support vertices in the solution instead of start from scratch and avoiding the leaf vertices being part of such solution. In order to do so, an analysis of the preliminary set of instances is shown in Table 2 to evaluate the percentage of vertices that are support vertices (column 2) and, additionally, the percentage of vertices that are dominated if support vertices are included in the solution (column 3).

Notice that, on average, almost 54% of the vertices belongs to the set of support vertices and, selecting those vertices result in 63.5% of the vertices dominated. This result shows the importance of considering support vertices. Analyzing every instance individually, in some cases, such as `as-22july06`, these percentages are high and the inclusion of support vertices in the solution almost completely solve the MDSP. On the contrary, there are some instances, such as `football`, in which there are not support vertices. Therefore, it is intelligent to maintain this idea of considering the support vertices to be part of the solution.

In IG framework, it is necessary to tune three parameters: the destruction and reconstruction methodology that will be considered, the percentage of destruction, and the stopping criterion. In this research, we follow a sequential experimental design, since it considerably reduces the number of experiments when comparing it with a full-factorial

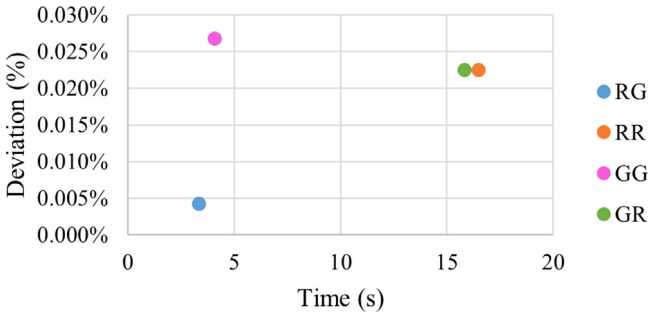


Fig. 5. Comparison of time and average deviation when considering the different combinations of destruction and reconstruction.

Table 3
Effect of different percentage of destruction in the Iterated Greedy algorithm.

β	Avg.	Time	Dev. (%)	#Best
0.1	739.78	**3.34**	0.93	8
0.2	**739.67**	4.85	**0.00**	**9**
0.3	739.78	7.60	0.93	8
0.4	**739.67**	12.10	**0.00**	**9**
0.5	739.78	15.26	0.93	8

design, obtaining similar results [26]. First of all, it is necessary to select the method used in the destruction and reconstruction phase: random or greedy destruction and random or greedy reconstruction. All combinations are tested fixing $\beta = 0.1$ and $\Delta = 250$. The rationale behind this is that we have followed an iterative experimental design, which usually leads to equivalent results as a full-factorial design [26]. Fig. 5 compares the average deviation with respect to the best solution found in the experiments, in percentage, with the computing time spend, in seconds.

In this comparison, it is desirable to be as close as possible to the origin of coordinates, since it would indicate that the method requires from small computing times and has a small deviation with respect to the best solution found in the experiments. Following this analysis, the best method is the *random destruction* and the *greedy reconstruction* (RG), since it is able to produce the best results in the shortest computing times. It is worth mentioning that considering the *random reconstruction* no matter if the destruction is random or greedy (GR and RR) will require from considerably larger computing times, producing very similar results between them. The reason behind this statement is that, when vertices are randomly added, the local search will spend more computational time improving the solutions after the destruction and reconstruction. Then, in the final algorithm, the destruction is performed following the random approach while the reconstruction is performed in a greedy way, balancing diversification and intensification.

The next experiment tries to evaluate the best value for the β parameter, which indicates the percentage of vertices destroyed from a minimal dominating set in each iteration. In particular, the values $\beta = 0.1, 0.2, 0.3, 0.4, 0.5$ are tested, while values larger than 0.5 are not considered since it would destroy more than half of the solution, being equivalent to have a multi-start approach. Table 3 shows the obtained results in this experiment.

As it can be observed, results are quite similar among them in terms of objective function value, although $\beta = 0.2$ and $\beta = 0.4$ seem to be the most promising values regarding the quality. Obviously, the computing time considerably increases with increasing β values. Therefore, comparing these two values, it seems reasonable to set the β value to 0.2, since it is the fastest one.

Finally, it is necessary to establish the value of Δ, which is the number of iterations without improvement allowed in IG metaheuristic and it is used as the stopping criterion. Table 4 shows the results obtained when considering $\Delta = \{50, 100, 150, 200, 250\}$.

In view of the results, it can be stated that the larger the value of Δ, the better the results and, obviously, the larger the computing time required. However, the algorithm converges when reaching $\Delta = 200$ and, therefore, this is the value fixed in the final configuration.

Table 4
Comparison of different values of Δ in the Iterated Greedy algorithm.

Δ	Avg.	Time	Dev. (%)	#Best
50	739.78	**1.96**	0.93	8
100	739.78	2.78	0.93	8
150	739.78	3.85	0.93	8
200	**739.67**	4.27	**0.00**	**9**
250	**739.67**	4.84	**0.00**	**9**

Table 5
Contribution of each part of the algorithm to the final design.

Algorithm	Avg.	Time	Dev. (%)	#Best
GIP	751.67	**0.13**	4.90	3
GIP + ELS	740.67	0.19	3.61	5
IG	**739.67**	4.28	**0.00**	**9**

The final preliminary experiment is not designed for configuring any parameter of the algorithm but for evaluating the contribution of each part of the proposed algorithm to the final version. Therefore, first results of the GIP to generate solutions are compared with the GIP coupled with the efficient local search (ELS) and, finally, with the complete IG algorithm. To summarize, IG algorithm generates an initial solution using the GIP, then a local search method is applied. Next, 20% of vertices are randomly removed in the destruction phase, and they are greedily added in the reconstruction phase until 200 iterations without improvements are reached, returning then the minimal dominating set found. Table 5 compares these three approaches.

As it can be derived from Table 5, IG algorithm is the most relevant part of the final design, and it does not considerably increase the computing time, requiring less than 5 s on average. It is important to remark that the local search is able to reduce the average percentage deviation and find a larger number of best solutions than the GIP marginally increasing the computing time. Finally, it is worth mentioning that the GIP isolated is able to produce high-quality solutions, being, on average, a 4.90% of the best solution found. This experiment proves the relevance of the complete IG framework.

3.2. Final experimentation

Having configured the complete algorithm, this Section is devoted to perform a competitive testing against the state-of-the-art methods, with the aim of evaluating the contribution of our proposal. To that end, two experiments are performed. The first one considers the same set of 28 instances used in [4]. The algorithms compared in this experiment are: the proposed Iterated Greedy, IG; an Order-based Randomized Local Search which is the state of the art, RLS; the implementation of the mathematical model using the mathematical programming solver Gurobi, ILP; a basic Ant Colony Optimization algorithm coupled with a local search, ACO-LS, by [22]; a variant of ACO-LS with a preprocessing phase, ACO-PP-LS, by [23]; and, finally, a new variant of ACO-LS which select vertices involved in each transition and initialize the pheromones according to the greedy results, ACO-LS-S.

It is worth mentioning that, with the aim of having a fair comparison, we have used exactly the same parameters for each algorithm as established in the original work. We refer the reader to [4] for a detailed explanation of each parameter and its value. Table 6 shows the obtained results. In all the results, the total computing time is considered, including the generation of the initial solution.

If we focus on RLS, ACO-LS, ACO-PP-LS and ACO-LS-S, the four algorithms proposed in the state-of-the-art, we can secure that ACO-LS and ACO-PP-LS are those that obtained worst results in terms of quality. Between ACO-LS-S and RLS, RLS reaches a smaller deviation and more number of best solutions, confirming that RLS is better than the other algorithms included in the comparison. Comparing IG and RLS, IG clearly outperforms RLS not only considering the objective function value but also the computational time, being almost 4 times faster on average. Results obtained suggest that the considered instances are not really challenging given that the mathematical programming solver (ILP) is able to solve all of them in less than a second, on average. This is mainly due to the inherent structure of the instances, which probably present a flat landscape in which several solutions reach the

Table 6
Comparison of the Iterated Greedy (IG), the Order-based Randomized Local search method (RLS), a mathematical programming solver (ILP), and three variants of Ant Colony Optimization (ACO-LS, ACO-PP-LS, and ACO-LS-S), when considering the same set of instances presented in [4].

Algorithm	Avg.	Time	Dev. (%)	#Best
IG	1304.68	164.38	0.04	20
ILP	**1303.11**	**0.25**	**0.00**	**28**
RLS	1309.35	600.00	0.16	20
ACO-LS	1550.66	600.00	11.82	13
ACO-PP-LS	1545.96	600.00	11.51	14
ACO-LS-S	1341.33	600.00	1.33	15

Table 7
Comparison of the Iterated Greedy (IG), the Order-based Randomized Local search method (RLS), a mathematical programming solver (ILP), and three variants of Ant Colony Optimization (ACO-LS, ACO-PP-LS, and ACO-LS-S), considering the newly generated set of 96 instances.

Algorithm	Avg.	Time	Dev. (%)	#Best
IG	**10.75**	**23.88**	**1.23**	**88**
ILP	11.94	1799.77	10.18	43
RLS	12.53	600.00	15.07	14
ACO-LS	14.12	600.00	33.19	0
ACO-PP-LS	14.11	600.00	33.17	0
ACO-LS-S	14.12	600.00	33.18	0

optimal value. This means that the minimal dominating set is not unique, being able to get more than one set with the same objective function value. In such situations, ILP is able to effectively stop the search having identified the optimal value, while the heuristic approaches are not able to identify that situations. Notwithstanding, the proposed IG is able to reach a slightly smaller deviation by requiring considerably smaller computing times.

With the aim of evaluating the three approaches in a set of more challenging instances, we generate a new testbed of 96 instances using the well-known Knuth's Algorithm S (see [17]) to generate a random set of instances which are more diverse and lack of structure. Table 7 shows the results obtained by the six approaches over this new set of instances.

As expected, this set of 96 more challenging instances makes ILP reach its limits, since it is not able to reach the optimal value in the time limit of 1800 in most of the instances. However, it is remarkable that ILP is able to reach the best known solution in 43 instances in that time limit, although it is not able to guarantee optimality. On the contrary, IG shows its performance by reaching 88 out of 96 best solutions with an average percentage deviation of 1.23%. The reduced value in this metric indicates that, even in those instances in which IG does not reach the best value, it still remains really close to it.

If we now focus on the best previous approach, RLS, it reaches 14 out of 96 best solutions with a deviation of 15.07%. Additionally, the computing time required by RLS is one order of magnitude larger than the current proposal, emerging IG as a competitive algorithm to address the MDSP in terms of both quality and computing time. Finally, the performance of the solver is also remarkable, being able to reach a smaller deviation and a larger number of best solutions than RLS.

The remaining algorithms of the literature, ACO-LS, ACO-PP-LS, and ACO-LS-S, show a similar performance in terms of quality and computing time, being the differences among them neglectable. These results highlight the difficulty of the newly generated instances, where only the best previous approach and the proposed algorithm are competitive for solving the MDSP in reasonable computing times.

Table 8
Ranking resulting from applying non-parametric Friedman statistical test. The obtained p-value is smaller than 0.01.

Algorithm	Mean rank
IG	1.40
ILP	2.24
RLS	2.88
ACO-LS	4.84
ACO-PP-LS	4.83

Table 9
Results obtained when applying the non-parametric Wilcoxon Signed Ranks Test considering all possible pairs derived from the best three approaches, obtaining a p-value smaller than 0.01 in all of them.

ILP–IG	ILP < IG	8
	ILP > IG	53
	ILP = IG	35
RLS–IG	RLS < IG	6
	RLS > IG	76
	RLS = IG	14
RLS–ILP	RLS < ILP	10
	RLS > ILP	63
	RLS = ILP	23

3.3. Analysis

The previous section shows the experimental results obtained when executing the proposed algorithm and the best methods found in the literature over the original set of instances and, also, over a more challenging set generated to evaluate the limits of the algorithms. This section is designed to further analyze the results obtained by the compared algorithms.

First of all, the well-known non-parametric Friedman test is applied, with the aim of ranking the compared algorithms. The resulting p-value smaller than 0.01 indicates that, as expected, there are statistically significant differences among them, resulting in the ranking presented in Table 8.

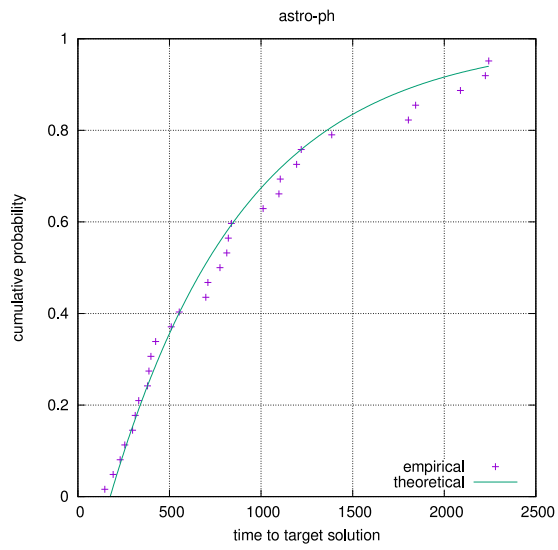
As it can be seen, the first algorithm in the ranking is clearly IG, followed by ILP and RLS, which are close to each other. Finally, the three ACO variants show similar performance, ranking in the last three positions.

Once it is confirmed that the three best algorithms are IG, RLS, and ILP, we now perform the pairwise non-parametric Wilcoxon statistical test to check if the null hypothesis that two of the samples belong to the same population can be rejected. In this case, since three algorithms are compared, we evaluate the three possible pairs derived, namely: IG–RLS, IG–ILP, and RLS–ILP. First of all, the proposed IG is compared with the best heuristic method RLS. Table 9 shows the results obtained in this test:

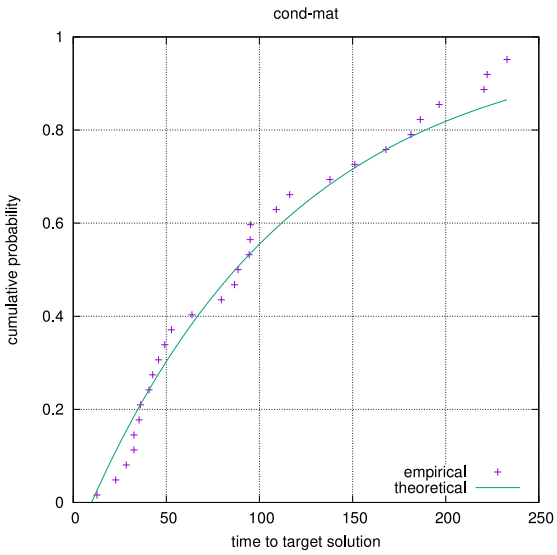
The resulting p-value smaller than 0.01 indicates that the null hypothesis is rejected, which indicates that IG is statistically better than ILP. Additionally, it can be seen that ILP is able to reach a better solution than IG only in 8 out of 96 instances, while IG outperforms the best solution reached by ILP (in those cases where the optimal value is not reached) in 53 out of 96.

Then, when considering IG versus the best heuristic approach, RLS, the null hypothesis is again rejected with the resulting p-value smaller than 0.01. Therefore, IG emerges as the best approach for solving the MDSP. In this case, RLS outperforms IG in just 6 cases while IG is better than RLS in 76 out of 96 instances.

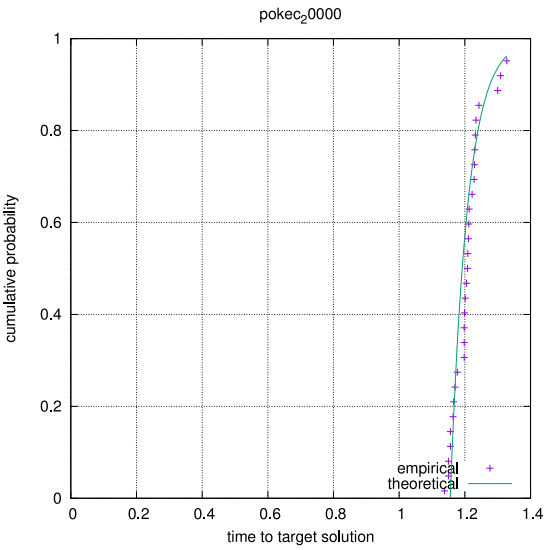
Finally, it is interesting to evaluate the best previous exact and heuristic approach. In this case, the resulting p-value smaller than 0.01 indicates rejects the null hypothesis, indicating that ILP is statistically better than RLS. Notice that this test does not consider the computing time, so ILP is only suitable for those cases in which the time is not a constraint.



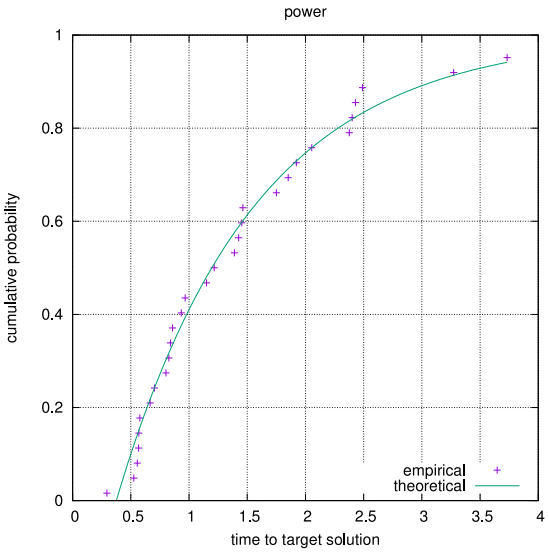
(a) astro-ph



(b) cond-mat



(c) pokec_20000



(d) power

Fig. 6. Time to target plots for four representative instances extracted from the used dataset.

After the discussion of results, it is interesting to analyze the probability of finding the best solution given a certain time horizon, to verify the robustness of the proposal. In order to do so, we use time-to-target plots. The hypothesis is that computing times fit a shifted exponential distribution. Given a certain instance, the algorithm is executed 30 times measuring the computing time required to reach the best solution found by the algorithm in each run. Then, the empirical and theoretical distributions are compared following the graphical methodology for data analysis described in [5]. For each instance, the computing times are ordered in an ascending order and then, each sorted running time is assigned a probability. Fig. 6 shows the cumulative probability distribution for each considered instance.

Fig. 6(a) shows the results when considering the instance `astro-ph`. In this case, it can be seen that, although it requires 2500 s approximately in the worst case to reach the best solution, it can reach it in 1500 s with a probability larger than 80%. In the case of `cond-mat`, depicted in Fig. 6(b), the maximum time required by IG is 250 s, but in 100 s it finds the best solution with a probability of 60%. The `pokec_2000` instance, shown in Fig. 6(c), is a particular case in which most of the runs require similar times, approximately 1.3 s, so there are not specific differences in terms of probability. Finally, the best solution for `power`, see Fig. 6(d), requires 4 s in the worst case, but in 2 s the probability of reaching the best solution is almost 80%. These results show the robustness of the method when considering 30 independent runs.

4. Conclusions

This paper solves an interesting problem, known as the Minimum Dominating Set Problem (MDSP), which models real situations appearing in many different real situations such as, the identification group of people being representatives for all the group, monitoring for a computer network, or influence maximization problem in social networks, among others. The Iterated Greedy is conformed by intuitive procedures to get an initial solution, an efficient local search method to improve it, and simple but intelligent strategies for destructing and reconstructing a solution during the search.

The proposed strategies have been proven to be efficient and effective over the set of 28 instances used in the state of the art. Additionally, new instances have been generated to test the limitations of the exact approach. The presented IG algorithm is able to reach the best values in almost all instances without requiring vast computational efforts, emerging as a competitive algorithm for solving the MDSP.

In the context of the design of metaheuristic algorithms, it is important to highlight the strengths and weaknesses of our proposal. The destruction phase allows the algorithm to escape from local optima, guiding the search to new and close promising regions of the search space, which will eventually lead to improvements. However, the method stagnates where the region explored is a basin of attraction and the global optimum is located in a region which is located far away from the region under exploration.

The knowledge of the MDSP allows us to include intelligent information in the design of the algorithm. First of all, the solution space is drastically reduced since support vertices must be mandatory included in the solution and leaf vertices are never considered in the solution. This is a significant reduction when the number of support and leaf vertices is large, which results in a positive effect in the efficiency of the algorithm. However, the number of leaf and support vertices are not necessarily large in certain types of graphs, therefore this idea will not have any effect in the quality of the generated solutions.

Furthermore, the efficient local search method is able to drastically reduce the number of necessary moves to reach a local optimum with respect to the naïve one. This optimization allows the algorithm to decrease the required computing time, thus being able to solve more complex instances. Although this reduction usually leads to high-quality solutions in reduced computing times, as it has been experimentally tested, it is worth mentioning that the local search may miss to perform some movements that are not originally promising but eventually results in the exploration of new promising regions of the search space.

As future work, it would be interesting to solve different variants of this class of combinatorial optimization problems that bring the theoretical model closer to real-life situations such as considering weights in the selected vertices/edges, or being resilient to failures, among others.

Acknowledgments

A. Casado and J. Sánchez-Oro are supported by the “Ministerio de Ciencia e Innovación, Spain”, Grant Ref. PID2021-125709OA-C22, and by “Comunidad de Madrid” and “Fondos Estructurales” of European Union with Grant Refs. S2018/TCS-4566, Y2018/EMT-5062.

S. Bermudo and A.D. López-Sánchez acknowledge support from the Junta de Andalucía, FEDER-UPO Research & Development Call, Spain, reference number UPO-1263769.

References

[1] Albert-László Barabási, Réka Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512.
[2] Alejandra Casado, Sergio Pérez-Peló, Jesús Sánchez-Oro, Abraham Duarte, A GRASP algorithm with tabu search improvement for solving the maximum intersection of k-subsets problem, J. Heuristics 28 (1) (2022) 121–146.
[3] Pedro Casas-Martínez, Alejandra Casado-Ceballos, Jesús Sánchez-Oro, Eduardo G. Pardo, Multi-objective grasp for maximizing diversity, Electronics 10 (11) (2021) 1232.
[4] David Chalupa, An order-based algorithm for minimum dominating set with application in graph mining, Inf. Sci. 426 (2018) 101–116.
[5] John M. Chambers, William S. Cleveland, Beat Kleiner, Paul A. Tukey, Graphical Methods for Data Analysis, Chapman and Hall/CRC, 2018.
[6] Aron Culotta, Maximizing Cascades in Social Networks: An Overview, 2003.
[7] C.F. De Jaenisch, Applications de L'analuse Mathematique an Jen des Echecs, Petrograd, 1862.
[8] Pedro Domingos, Matt Richardson, Mining the Network Value of Customers, KDD '01, Association for Computing Machinery, New York, NY, USA, 2001, pp. 57–66.
[9] D. Erwin, Dominating broadcasts in graphs, Int. J. Comput. Aided Eng. Technol. 42 (2004) 89–105.
[10] Fedor V. Fomin, Fabrizio Grandoni, Dieter Kratsch, A measure & conquer approach for the analysis of exact algorithms, J. Assoc. Comput. Mach. 56 (5) (2009) 1–32.
[11] Pierre Hansen, Nenad Mladenović, First vs. best improvement: An empirical study, Discrete Appl. Math. 154 (5) (2006) 802–817.
[12] Teresa W. Haynes, Sandra M. Hedetniemi, Stephen T. Hedetniemi, Michael A. Henning, Domination in graphs applied to electric power networks, SIAM J. Discrete Math. 15 (4) (2002) 519–529.
[13] Teresa W. Haynes, S.T. Hedetniemi, Peter J. Slater, Fundamentals of Domination in Graphs, Marcel Dekker, New York, 1998.
[14] Teresa W. Haynes, S.T. Hedetniemi, Peter J. Slater, Domination in Graphs: Advanced Topics, Marcel Dekker, New York, 1998.
[15] Abdel-Rahman Hedar, Rashad Ismail, Hybrid genetic algorithm for minimum dominating set problem, in: ICCSA, 2010.
[16] D. Kempe, J.M. Kleinberg, É. Tardos, Influential nodes in a diffusion model for social networks, in: L. Caires, G.F. Italiano, L. Monteiro, C. Palamidessi, M. Yung (Eds.), ICALP, in: Automata, Languages and Programming, vol. 3580, Springer, Berlin, Heidelberg, 2005, pp. 1127–1138.
[17] Donald E. Knuth, The Art of Computer Programming, Vol. 1: Fundamental Algorithms, third ed., Addison-Wesley, Reading, Mass., 1997.
[18] Wenhan Li, Xiaolong Chen, Junqing Li, Hongyan Sang, Yuyan Han, Shubo Du, An improved iterated greedy algorithm for distributed robotic flowshop scheduling with order constraints, Comput. Ind. Eng. 164 (2022) 107907.
[19] Ana Dolores López-Sánchez, Jesús Sánchez-Oro, Alfredo García Hernández-Díaz, GRASP and VNS for solving the p-next center problem, Comput. Oper. Res. 104 (2019) 295–303.
[20] O. Ore, Theory of Graphs, AMS, Providence, 1962.
[21] Sergio Pérez-Peló, Jesús Sánchez-Oro, Ana Dolores López-Sánchez, Abraham Duarte, A multi-objective parallel iterated greedy for solving the p-center and p-dispersion problem, Electronics 8 (12) (2019).
[22] Anupama Potluri, Alok Singh, Two Hybrid Meta-heuristic Approaches for Minimum Dominating Set Problem, 2011, pp. 97–104.
[23] Anupama Potluri, Alok Singh, Hybrid metaheuristic algorithms for minimum weight dominating set, Appl. Soft Comput. 13 (2013) 76–88.
[24] Juan David Quintana, Raul Martin-Santamaria, Jesus Sanchez-Oro, Abraham Duarte, Solving the regenerator location problem with an iterated greedy approach, Appl. Soft Comput. 111 (2021) 107659.
[25] Ruben Ruiz, Thomas Stutzle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, European J. Oper. Res. 177 (3) (2007).
[26] Jesús Sánchez-Oro, Manuel Laguna, Rafael Martí, Abraham Duarte, Scatter search for the bandpass problem, J. Global Optim. 66 (4) (2016) 769–790.
[27] Jesús Sánchez-Oro Calvo, Abraham Duarte, Iterated greedy algorithm for performing community detection in social networks, Future Gen. Comput. Syst. 88 (2018).
[28] Laura A. Sanchis, Experimental analysis of heuristic algorithms for the dominating set problem, Algorithmica 33 (2002) 3–18.
[29] Peng Sun, Xiaoke Ma, Dominating communities for hierarchical control of complex networks, Inf. Sci. 414 (2017) 247–259.
[30] Haynes Teresa W., Hedetniemi Stephen T., Henning Michael A., Structures of Domination in Graphs, Springer Nature, 2021.
[31] Peng-Jun Wan, K.M. Alzoubi, O. Frieder, Distributed construction of connected dominating set in wireless ad hoc networks, in: Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, Vol. 3, 2002, pp. 1597–1604.