LII Simpósio Brasileiro de Pesquisa Operacional
João Pessoa - PB, 3 a 5 de novembro de 2020

Metaheuristics for the Weighted 2-Dominating Set Problem in Graphs

André Felipe de Souza Mota, Eugenio Belizário Ribeiro Faria, Stênio Sã Rosário Furtado Soares
UFJF - Universidade Federal de Juiz de Fora
Campus Universitário, Rua José Lourenço Kelmer, s/n - São Pedro, Juiz de Fora - MG
andrefelipe@ice.ufjf.br, eugenio.faria@ice.ufjf.br, ssoares@ice.ufjf.br

ABSTRACT

Given a graph $G = (V, E)$, a function $f : V \to \mathbb{Z}$ and positive integer k, a set $D \subseteq V$ is a k-domination if $\forall\, v \in V$, $v \in D$ or $N[v] \cap D \geq k$. The problem of minimum weight k-domination is an NP-hard problem and consists of obtaining a k-domination D such that $\sum_{u \in D} f(u)$ is minimal. This paper presents three approaches for the minimum weight k-domination problem with $k = 2$: two evolutionary approaches, one being a Genetic Algorithm with local search and the other a Memetic Algorithm; besides a Simulated Annealing approach. The experiments considered a set of 1020 instances from the literature. Using the results obtained from the linear problem solver Gurobi, it was possible to show the convergence capacity of the proposed approaches for optimal solutions or near-optimal.

KEYWORDS. Metaheuristics, 2-Dominanting Set, Graphs.

Topics (Genetic algorithm, Memetic algorithm, Simulated annealing, Constructive algorithm, Local search heuristics)

Galoá { Este trabalho foi publicado utilizando o **Galoá** proceedings

1. Introduction

The minimum weight dominating subset problem (MWDSP) is broadly studied, with applications in many fields (Qian et al. [2015]; Shen e Li [2010]; Wang et al. [2018]; Zhu et al. [2012]). In Qian et al. [2015], the authors presented the problem as an especial case of the classic set cover problem for obtaining smaller subsets of variables in the Pareto frontier for applications in machine learning. In the same way, considering the problem as an especial case of set cover, Gao et al. [2014] proposed an heuristic approach based in local search and perturbation applied to the Steiner Triple Systems problem.

The concept of k-domination in graphs was first introduced by Fink e Jacobson [1985], in which was proved that the decision version of the problem is NP-complete for general graphs. Considering a simple undirected graph, that is, without multi-edges or self-loops, $G = (V, E)$ and a positive integer k, the optimal version of the problem consists in obtaining a smallest cardinality set $D \subseteq V$ such that every vertex $v \in V \setminus D$ is adjacent to at least k vertexes of D. In Lan e Chang [2013] the authors introduced an linear time algorithm for the problem considering graphs with specific topologies, like circles, trees, cliques blocks etc. Chiarelli et al. [2018] presented algorithms for the k-domination problem in proper interval graphs [1] with complexity $O(n^{3k})$.

Befits highlighting that a variation of the k-domination problem found in the literature is related to the number of edges between the dominating set nodes and that dominated, not being covered in this paper. In this problem, the positive integer k is related to the minimum edges necessary to connect any dominated node to some dominating vertex. In which case, we have that $D \subseteq V$ is a k-dominating set if every node $u \in V \setminus D$ is in a distance of at least k edges of any vertex $v \in D$. This problem has usage in social media monitoring and in the sensors networks project Nguyen et al. [2019].

Since the applications of the problem in general involve costs associated to the implantation of some sort of service or functionalities in the graph nodes, the problem version that deal with node-weighted graphs is of great theoretical and practical interest (Ping Wu et al. [2006]; Zhu et al. [2012]; Wang et al. [2017]; Shen e Li [2010]). In this case, the problem consists in to obtain a dominating set D whose sum of the nodes weights is minimal. A thorough study of the dominating set problem variations is presented in Mayra Carvalho Albuquerque [2018]. In Albuquerque e Vidal [2018] the authors presents an approach that combines exact methods with meta-heuristics for the weighted dominating set problem.

The unweighted 2-dominating subset problem has been addressed in Huang et al. [2007], where a self-stabilizing algorithm was proposed. In the other hand, in this paper we consider the node-weighted version and $k = 2$. Let $G = (V, E, f)$ an undirected node-weighted graph, where V is the vertex set, E is the edge set, and $f : V \to \mathbb{Z}$. Obtain a set $D \subseteq V$ such that $\forall\, u \in V \setminus D$ exists at least two neighboring nodes to u in D and $\sum_{u \in D} f(u)$ is minimal. The problem can be model as a linear programming problem. Thereby, consider: $V = \{1, \ldots n\}$ the graph nodes set; $N[i]$: vertex i closed neighborhood; $f(i)$: vertex i weight; x_i: binary variable that assumes value 1 if the vertex i is in the solution or 0, otherwise. Then, the problem is given by:

$$\text{Minimize } z = \sum_{i \in V} f(i) x_i \tag{1}$$

[1] An interval graph is an undirected graph obtained from a set of intervals where each vertex is an interval and an edge between two of them indicates that there is an intersection between them. When any interval contain totally the other, the graph is said a proper interval.

Subejct to:

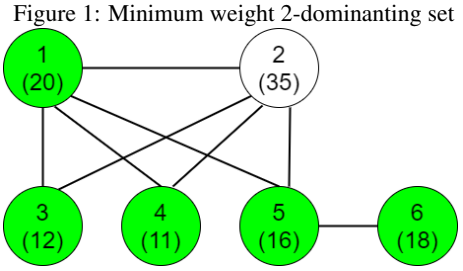
$$\sum_{j \in N[i]} x_j \geq 2 \quad \forall i \in V \tag{2}$$

$$x_i \in \{0, 1\} \quad \forall i \in V \tag{3}$$

The constraints set (2) assures that every vertex outside of the solution has at least two vertex of its closed neighborhood within the solution. Note that, once the neighborhood is closed, any vertex in the solution demands only one other adjacent vertex to be in the solution as well. The domain of the variables is define in the constraints 3.

As shown in Figure 1, where the nodes weight is in parentheses, the nodes 3, 4 and 5 can be dominated by 1 and 2. As the vertex 6 needs to be 2-dominated and only has one neighbor, both 5 and 6 nodes must be in any solution. Therefore, the 2-dominating set with the minimum nodes is $\{1, 2, 5, 6\}$. However, for the weighted version, this would not be the best solution, once the vertex 2 weight considerably impact the final cost, whose value is 89, so that, although dominating 3 and 4, including those nodes in the solution instead of vertex 2 benefits the quality of the solution. Thus, the solution $\{1, 3, 4, 5, 6\}$, represented in green in the figure, would have a weight of 77 and would be optimal for such a graph.

Figure 1: Minimum weight 2-dominanting set



As the classic dominating set problem, the k-domination problem is NP-hard. So, the use of heuristics approaches is common in the literature. In this paper, three approaches are proposed to solve the problem and establish a comparison between them, in order to identify the capacity of each technique to obtain competitive solutions when comparing the results with those obtained from the use of a solver.

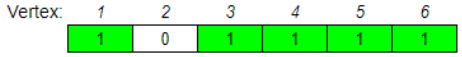
The remaining paper is organized as follows: Section 2 features the detailing of the proposed algorithms; Section 3 describes the computational experiments performed and presents the results analysis, while Section 4 shows the conclusions and indicates possible work lines.

2. Proposed approaches

In the process of development of the three meta-heuristics proposed for the problem, we first sought to analyze different construction and local search strategies, in order to identify possible bottlenecks related to the characteristics of the problem, such as the edge density of the graph, the weight distribution of the nodes, and so on. In this section, the construction and local research algorithms are presented, in addition to the three proposed meta-heuristics.

To represent a solution, we consider a binary vector whose size is defined by number of nodes in the instance, where the value 1 in the position i indicates that the node $i \in D$. Figure 2 show the representation of the ideal solution to the graph of Figure 1.

Figure 2: Solution representation



2.1. Constructive Algorithm

To build basic solutions that will later be refine by the meta-heuristics, a randomized greedy algorithm was developed, in which, from an empty solution, at each iteration a vertex is chosen from among the best, according to a selection criterion, and added to the partial solution. The vertex, once placed in the solution, will never be removed, that is, there is no review of decisions already taken. The procedure is repeated until the solution becomes viable, which means that all the nodes are 2-dominated. Since the choice is not entirely greedy, is possible to get a variety of solutions, since the best candidate is not chosen, but one of the best candidates.

The algorithm 1 shows the proposed constructive method pseudo-code. It receives as parameters a value α, that defines the percentage of the best elements ranked according to the criterion established in the parameter λ and the input graph $G = (V, E)$. The function *heuristcSelection()* selects one of the $\alpha\%$ best nodes of the candidates set based on the pre-established criterion λ. In order to allow that different regions of the solution space are explored, six different criteria were used in ranking candidates, of which five were brought from Mayra Carvalho Albuquerque [2018] for classic weighted dominating set problem ($\lambda_1, \lambda_2, \lambda_3, \lambda_4$ e λ_5), to know: λ_1: lower weight; λ_2: upper number of neighbors not yet dominated; λ_3: biggest ratio between the number of non-dominated neighbors and the vertex weight; λ_4: biggest ratio between the weights sum of the non-dominated neighborhood and the vertex weight; λ_5: biggest ratio between the product of the weights sum of the non-dominated neighborhood and the number of non-dominated neighbors divided by its weight; furthermore the proposed criterion λ_6: biggest weights sum of the non-dominated neighborhood.

Algorithm 1 Greedy randomized

```
1: Input: α, λ, G
2: S ← {}; /* S is the vertex set of the solution */
3: C ← sortVertex(G(V), λ);
4: while S not viable do
5:     x ← heuristicSelection(C, λ, α);
6:     S ← S ∪ x;
7:     C ← C − x;
8: end while
9: return S
```

2.2. Local Search

Local search heuristics go through solution neighborhoods in an attempt to find a better solution than the current one. The solution s neighborhood consists of a finite set solutions obtained from changes made to the structure of s defined by an operation or movement. Each movement defines a neighborhood and, once all the solutions in the same have been verified, the best solution found is called local optimal for the neighborhood of s.

Since a given movement can lead to solutions that violate the constraints of the problem, the generated solutions can be submitted to some feasibility procedure, or simply be discarded. In this paper, only viable solutions were considered. The use of different local search heuristics for a

given solution s allows different regions closes to s to be intensified in the search process. A technique widely explored in the literature for different optimization problems when having different movements is the variable neighborhood descendent (VND) method, first proposed by Mladenović e Hansen [1997], in which a sequence of local searches is performed, modifying the movement performed during the search.

Thus, to compose the VND two movements were used: remove one of the nodes from the current solution (**del**) and delete a node from the current solution and add another (**swap**). It is worth mentioning that, if there is no better neighbor, the functions **del** and **swap** return the initial solution. In addition, the **first** improvement and **best** improvement updates were evaluated. The operation of the proposed local search is described in the Algorithm 2.

Algorithm 2 Local Search

1: **Input:** initialSolution S, first or best improvement
2: $movement \leftarrow [del(), swap()]$;
3: $k \leftarrow 0$;
4: **while** $k < 2$ **do**
5: $S^* \leftarrow movement[k](S)$;
6: **if** $weight(S^*) < weight(S)$ **then**
7: $k \leftarrow 0$;
8: $S \leftarrow S^*$;
9: **else**
10: $k \leftarrow k + 1$;
11: **end if**
12: **end while**
13: **return** S^*;

2.3. *Simulated Annealing*

The Simulated Annealing (SA) approach is a meta-heuristic first proposed by Kirkpatrick e Gelatt Jr [1983] that mimics a thermodynamic procedure, where the potential energy of matter is minimized through a slow cooling process after being subjected to high temperature. In this technique, the material undergoes controlled heating and cooling, so that the material's atoms gain energy to move freely in the heating phase and, when cooling, they reorganize in a configuration with less internal energy, resulting in a increase in the material crystal size and, consequently, less defects.

The SA algorithm starts with a high temperature and a initial solution s. Throughout the process, the solution can be replaced by a neighbor solution with probability of $1 - e^{\frac{-(\Delta E)}{T}}$, being ΔE the difference between the energy of the obtained neighbor solution and the current solution. Thus, the higher the current temperature value, the greater the chance that a worse solution will be accepted. As the algorithm progresses through iterations, the temperature value is decremented, until it is low enough that no neighbor solution worse than the current solution is accepted.

The Algorithm 3 shows the Simulated Annealing pseudo-code developed for the 2-dominating set. The function *greedyRandomized* in line 2 builds the initial solution, as described in Section 2.1, choosing randomly between criteria 1, 2, 4, 5 or 6, once criterion 3 was discarded because it did not presented good results when combined with the SA. The, the function *localSearch* in line 3 performs a local search in the solution, as described in the Section 2.2, choosing randomly between

first improvement or best improvement. Furthermore, the function *randomNeighbor* in line 8 returns a viable random neighbor of the current solution, while the function *weight* in line 9 returns the weight sum of the nodes in the solution. To form this neighbor, first it tries to remove a random vertex from the current solution and, if a viable neighbor is not found, it tries to remove a random vertex and add another. If a viable neighbor still is not found, another node will be added to the solution.

Algorithm 3 Simulated Annealing

1: **Input:** startTemp T_i, finalTemp T_f, tempInterations $iter$, coolingRate β;
2: $s \leftarrow randomizedGreedy(\alpha, randomCriterion)$;
3: $s \leftarrow localSearch(S, randomCriterion)$;
4: $s^* \leftarrow S$;
5: $T \leftarrow T_i$;
6: **while** $T > T_f$ **do**
7: $amt \leftarrow 0$;
8: **while** $amt < iter$ **do**
9: $s' \leftarrow$ randomNeighbor(s);
10: $\Delta E \leftarrow$ weigth(s') - weight(s);
11: **if** ΔE ≤ 0 **then**
12: $s \leftarrow s'$;
13: **if** weigth(s) $<$ weigth(s^*) **then**
14: $s^* \leftarrow s$;
15: **end if**
16: **else if** $random(0,1) > e^{\frac{-\Delta E}{T}}$ **then**
17: $s \leftarrow s'$;
18: **end if**
19: $amt \leftarrow amt + 1$;
20: **end while**
21: T$\leftarrow$ T$\times\beta$;
22: **end while**
23: **return** s^*

2.4. Genetic Algorithm

The Genetic Algorithm (GA) was first introduced by John [1992] inspired in concepts of Darwin's theory of evolution. Classified as population based meta-heuristic, it is an iterative procedure that performs the population evolution over generations through selection operators, recombination or crossover and mutation, processes presents in the evolution of some species. The Algorithm 4 shows the proposed Genetic Algorithm with local search pseudo-code. Each solution or chromosome of the initial population P_0 is obtained from the randomized greedy algorithm drawing one of the six criteria presented in Section 2.1 with equal probability. Each generation t, a new population P_t is obtained with 10% of the generation P_{t-1} best individuals, 80% of viable offspring obtained from selection operators, crossover and mutation, and 10% of new solutions obtained from the constructive algorithm.

The evolution process starts with the selection of parent solutions that will generate new individuals. This step is performed by *tourneySelection* in line 10, which consists of drawing two individuals from the population and the fittest between the two becomes a parent. After selecting

the parents, chances are that two offsprings will be generated by double crossover, which draws two indexes that will define the cut points in the solution structure. Then, each offspring can mutate. The mutation operator used in this paper is to invert two bits of the solution. If the resulting offspring is viable, he will become part of the new population. Lastly, with each generation, *eliteLocalSearch* in line 22 apply first improvement local search, described in the Section 2.2, in the 10% best individuals of the population.

Algorithm 4 Genetic Algorithm

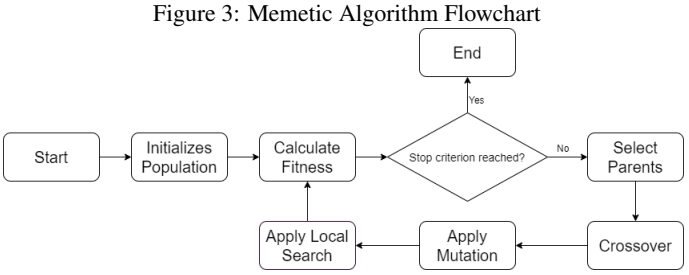
1: **Input:** $populationSize$, $numberOfGenerations$, $mutationRate$, $crossoverRate$
2: $pop \leftarrow startPopulation$;
3: $s^* \leftarrow pop[0]$;
4: **for** $i = 0; i < numberOfGenerations; i$++ **do**
5: **for** $j = 0; j < 0.1 * populationSize; j$++ **do**
6: $newPop[j] \leftarrow pop[j]$;
7: **end for**
8: $count \leftarrow 0.1 * populationSize$;
9: **while** $count < 0.9 * populationSize$ **do**
10: $tourneySelection(pop, parent1, parent2)$;
11: $crossoverChance \leftarrow random(0, 1)$;
12: **if** $crossoverChance \leq crossoverRate$ **then**
13: $doubleCrossover(parent1, parent2, offspring1, offspring2)$;
14: $mutation(mutationRate, offspring1, offspring)$;
15: $insertNewPopulation[cont](newPop, offspring1, offspring2)$;
16: **end if**
17: **end while**
18: **for** $j = 0.9 * populationSize; j < populationSize; j$++ **do**
19: $pop[j] \leftarrow greedyRandomized(\alpha, randomCriterion)$;
20: **end for**
21: $pop \leftarrow sort(pop)$;
22: $pop \leftarrow eliteLocalSearch(0.1, pop)$;
23: **if** weigth($pop[0]$) < weigth (s^*) **then**
24: $s^* \leftarrow pop[0]$;
25: **end if**
26: **end for**
27: **return** s^*;

2.5. Memetic Algorithm

Proposed by Moscato et al. [1989], the Memetic Algorithm is an extension of the Genetic Algorithm. Its differential is submitting the generated offspring to an intensification process, generally a local search algorithm, able to make improvements in the quality of solutions that the genetic operators cannot do. In the proposed algorithm, such behavior is obtained by performing first improvement local search in the solutions generated by the evolutionary process.

The flowchart in Figure 3 shows the Memetic Algorithm operation, in which the local search is applied right after the mutation stage. The other steps of this procedure were detailed in Section 2.4 and can be seen in the Algorithm 4. It is worth mentioning that, as there is no need, the function *eliteLocalSearch* is not used in this approach.

Figure 3: Memetic Algorithm Flowchart



3. Computational experiments

To verify the proposed approaches performance regarding the solution quality obtained and the processing time, a test set was performed for instances of different characteristics. For such, we sought to identify a parameter calibration that best suited each of the approaches. In this section, the results are summarized and presented.

A set of 1020 instances obtained from Romania [2010] was submitted to the the proposed algorithms. The instances were organized into two groups of 510 each. In instances of type **T1**, the nodes weight are evenly distributed over the interval between 20 and 70, while the ones of type **T2**, the weight of each vertex i is a random value between 1 and d_i^2, where d_i is the vertex degree. Within each of these types, instances are grouped into 51 groups of 10 instances, so that an instance named "Problem.dat_50_100_0" is the first of 10 instances that have 50 nodes and 100 edges. The instances of the group "Problem.dat_800_0" were not used, considering that they do not apply to the problem addressed in this paper, besides that, the results for the groups "Problem.dat_800_5000" and "Problem.dat_800_10000" could not be shown due to the lack of time to conclude the experiments.

All algorithms ran in a computer with i5-8250U CPU with 1.6 GHz (3.4 GHz turbo) clock, 8GB of RAM DDR4 2400 MHz, Linux operational system Ubuntu 20.04 LTS distribution. The programming language used was C++, compiler g++ (Ubuntu 9.3.0-10ubuntu2) 9.3.0. The mathematical model submitted to the solver Gurobi was developed in the language Python 2.7.

3.1. Parameters calibration

To tuning the parameters we used the Iterated Race (IRACE) framework, developed by López-Ibáñez et al. [2016], in which a scan is performed between candidate parameters to select the best setting, in addition to empirical tests, done to determine limits for the parameters. The exceptions are the following parameters, which were determined only through empirical tests:

- For the constructive algorithm, empirical tests were done with $\alpha \in \{10\%, 15\%, \ldots, 30\%\}$ and, regardless of which criteria were used, the best results were obtained from $\alpha = 10\%$, thus eliminating calibration by IRACE.
- For the SA, the start temperature parameter was adjusted through empirical tests with 1000 as start temperature, 10^{-3} as final temperature and 100 iterations per temperature, the average temperature being selected at which the acceptance rate of worse solutions was 95% (405.13). The final temperature was set at 10^{-3}.

The parameters of each algorithm, the options given to IRACE and the chosen parameter can be seen in Table 1. To compare the perform and results of the approaches developed, the mathematical model presented in Section 1 was submitted to the Gurobi 8.1.1 Optimization [2019] mathematical solver. For each one of 1020 instances, a time limit of 3600 seconds has been set. Gurobi runs several linear programming algorithms simultaneously, returning the result of the first one to finish and, for that, it can use all the cores of the execution environment.

Table 1: Parameters calibrated by IRACE

Algorithm	Parameter	Candidates	Selected
SA	Temperature iterations	100, 200, 300, 400	300
	Cooling rate	85%, 90%, 95%, 98%	98%
Genetic	Population size	50, 100, 150, 200, 250	200
	Number of generations	500, 1000, 1500, 2000	1500
	Mutation rate	20%, 30%, 40%	40%
	Crossover rate	70%, 80%, 90%	90%
Memetic	Population size	50, 100, 150, 200, 250	250
	Number of generations	250, 500, 750, 1000	1500
	Mutation rate	20%, 30%, 40%	20%
	Crossover rate	70%, 80%, 90%	70%

3.2. Results analysis

Since no benchmarks results are known for the weighted 2-dominating, the proposed approaches are compared with the solutions obtained by the Gurobi solver, which, out of a total of 1020 instances, manage to prove optimality in 865, of which 377 were type 1 and 488 type 2.

Each approach ran 30 times for each instance and cost and processing time were taken. Among the three proposed approaches, the GA was able to obtain a solution equal the optimum in 108 type 1 instances and in 176 type 2 instances, while the SA managed to achieve 153 in type 1 and 120 in type 2. In contrast, the memetic algorithm achieved results equal to that of the solver in 318 type 1 instances and in 379 type 2 instances, in addition to improving the result in 66 instances.

Although the SA algorithm has been shown less effective in relation to the memetic algorithm, it requires much less processing time. Besides that, when compared to the proposed AG with local search, the SA is a good alternative in terms of time/cost.

In order to analyze the algorithms performance regarding the graph density (ratio between the total graph edges and the edges number of a same order complete graph), the graphic shown in the Figure 4 shows the instances in increasing order of density on the abscissa axis and the percentage difference in relation to the best solution on the ordinate axis. Contiguous lines refer to type 1 instances and dashed lines to type 2. Since the graph topology is the same for both types, changing only the weight assignment, it is possible to evaluate both the impact of the different weight assignments and the density.

It can be seen in the Figure 4 that the least sensible approach to the graph density is the memetic algorithm regardless of type 1 or 2. It is also noticed that SA is the approach that best adapted to the T2 instances, demonstrating that the weight distribution that considers the square of the degree vertex implied a lower increase in the instance difficulty than verified in the other algorithms. In addition, for higher density instances, GA with local search significantly improves performance when compared to SA, especially in T2 instances.

The Table 2 presents a comparative summary of the proposed approaches experiments. For each type of instance, the **Best Weight** column shows the average best value of 510 entries grouped by vertex number, which is used by **AvgBest** and **AvgAll** to present the percentage deviation regarding the best results and all the executions of each approach, respectively. Besides that, the processing times demanded by each algorithm are exhibited.

It is easy to see that the proposed memetic algorithm, in addition to obtaining the best solution, the percentage deviation, in the AvgAll column, indicates that it is more robust in terms

Figure 4: Approaches performance according to graph density and instance type

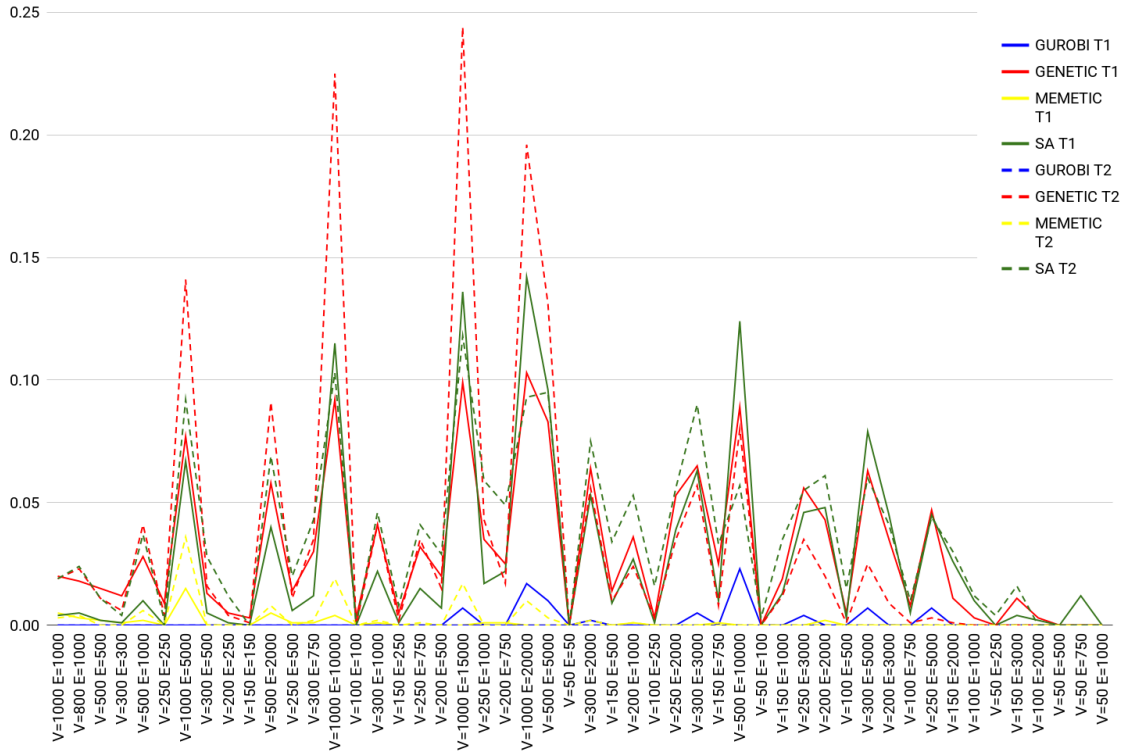


Table 2: Approaches results compiled by instance vertex number

INSTANCE		BEST	GENETIC			MEMETIC			SA		
T	|V|	WEIGHT	AvgBest	AvgAll	T(s)	AvgBest	AvgAll	T(s)	AvgBest	AvgAll	T(s)
1	50	423.20	**0.00%**	0.36%	**1.11**	**0.00%**	**0.00%**	7.92	0.05%	0.75%	1.32
1	100	849.07	0.36%	2.06%	**2.73**	**0.00%**	0.00%	16.16	0.18%	0.89%	4.46
1	150	1387.63	0.91%	2.91%	**5.72**	**0.00%**	0.07%	40.61	0.47%	1.72%	6.63
1	200	1789.38	1.93%	4.10%	10.04	**0.00%**	0.19%	80.18	1.37%	3.21%	**9.31**
1	250	2325.13	2.31%	4.11%	16.17	**0.00%**	0.35%	140.06	1.17%	2.86%	**12.07**
1	300	3090.41	2.65%	4.22%	24.19	**0.00%**	0.51%	214.32	1.54%	3.16%	**13.92**
1	500	5080.82	3.09%	4.45%	67.77	**0.00%**	0.73%	700.72	1.97%	3.62%	**25.77**
1	800	13045.60	2.36%	3.12%	213.92	**0.00%**	0.51%	2453.54	0.74%	1.55%	**26.28**
1	1000	7478.24	4.17%	5.43%	242.01	0.04%	1.10%	3221.40	3.74%	5.87%	**65.55**
1			1.98%	3.42%	64.85	0.00%	0.38%	763.88	1.25%	2.62%	**18.37**
2	50	309.35	**0.00%**	0.08%	1.06	**0.00%**	**0.00%**	7.37	0.11%	4.15%	**0.79**
2	100	729.72	0.06%	1.46%	2.93	**0.00%**	0.01%	16.98	0.86%	13.57%	**2.67**
2	150	1049.83	0.47%	3.92%	6.22	**0.00%**	0.03%	38.93	2.49%	21.84%	**4.47**
2	200	1390.10	1.53%	6.85%	11.37	**0.00%**	0.21%	81.18	4.48%	31.52%	**6.47**
2	250	1824.60	2.42%	8.66%	18.21	**0.00%**	0.31%	134.71	4.53%	35.36%	**8.37**
2	300	2033.70	3.72%	10.53%	26.63	**0.00%**	0.58%	204.16	5.81%	38.80%	**10.16**
2	500	3703.90	8.18%	16.04%	70.86	**0.00%**	1.19%	632.58	5.97%	51.07%	**18.62**
2	800	3415.20	3.88%	5.81%	224.71	**0.00%**	1.47%	1939.33	2.95%	14.33%	**22.73**
2	1000	9381.20	17.45%	28.61%	252.78	**0.00%**	2.96%	3185.07	7.82%	74.84%	**51.69**
2			4.19%	9.11%	68.31	**0.00%**	0.75%	693.37	3.89%	31.72%	**14.00**

of average standard deviation, however with time more than 100 times above the other evolutionary approach time.

It is also noticeable in Table 2 that the SA algorithm showed better results than the GA

with local search, having an average time around 9 times lower in T1 instances and 3 times T2 instances. However, the values of the respective AvgAll columns show that the evolutionary approach is more effective in exploring the search space.

4. Conclusions

This paper proposed a comparative analysis of three meta-heuristic for the minimum weight 2-dominating subset problem. Taking the results presented by the Gurobi solver as a reference, it was possible to verify the capacity of the proposed memetic algorithm to obtain solutions equal to the optimal ones and, in some cases, improve the solution when the solver did not close the gap. In addition, it was found that, with an average time around 37.46 times less than the average time of the memetic algorithm, SA obtained solutions with an average cost just 3.54% higher.

The performance analysis of the approaches according to the instances characteristics allowed to show that the nodes weight distribution influences differently in the difficulty found by each approach, as well as the input graph density.

As future works lines, a study on the variation of the k value and the problem complexity reduction are points of interest. Also, as the applications for the problem variations are known, searching for real instances and analyzing the response time constraints for the application can open new challenges about the problem. In the same work line, an improvement of SA can be done using more refined enhancement strategies, such as the use of a local search with other movements, that could result in competitive results while keeping an execution low time.

References

Albuquerque, M. e Vidal, T. (2018). An efficient matheuristic for the minimum-weight dominating set problem. *Applied Soft Computing*, 72:527 – 538. ISSN 1568-4946. URL http://www.sciencedirect.com/science/article/pii/S1568494618303922.

Chiarelli, N., Hartinger, T. R., Leoni, V. A., Pujato, M. I. L., e Milanic, M. (2018). New algorithms for weighted k-domination and total k-domination problems in proper interval graphs. *CoRR*, abs/1803.04327. URL http://arxiv.org/abs/1803.04327.

Fink, J. F. e Jacobson, M. S. (1985). *N-Domination in Graphs*, p. 283–300. John Wiley Sons, Inc., USA. ISBN 0471816353.

Gao, C., Weise, T., e Li, J. (2014). A weighting-based local search heuristic algorithm for the set covering problem. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, p. 826–831.

Huang, T. C., Lin, J.-C., Chen, C.-Y., e Wang, C.-P. (2007). A self-stabilizing algorithm for finding a minimal 2-dominating set assuming the distributed demon model. *Computers Mathematics with Applications*, 54(3):350 – 356. ISSN 0898-1221.

John, H. (1992). Holland. genetic algorithms. *Scientific american*, 267(1):44–50.

Kirkpatrick, S. e Gelatt Jr, C. (1983). and mp vecchi. *Optimization by simulated annealing. Science*, 220(4598):671–680.

Lan, J. K. e Chang, G. J. (2013). Algorithmic aspects of the k-domination problem in graphs. *Discrete Appl. Math.*, 161(10–11):1513–1520. ISSN 0166-218X. URL https://doi.org/10.1016/j.dam.2013.01.015.

Galoá { Este trabalho foi publicado utilizando o **Galoá** proceedings

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

Mayra Carvalho Albuquerque, T. V. G. V. (2018). *Matheuristics for Variants of the Dominating Set Problem*. PhD thesis, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

Mladenović, N. e Hansen, P. (1997). Variable neighborhood search. *Comput. Oper. Res.*, 24(11): 1097–1100. ISSN 0305-0548. URL https://doi.org/10.1016/S0305-0548(97)00031-2.

Moscato, P. et al. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989.

Nguyen, M. H., Hà, M. H., Hoang, D. T., Nguyen, D. N., Dutkiewicz, E., e Tran, T. T. (2019). An efficient algorithm for the k-dominating set problem on very large-scale networks (extended abstract). In Tagarelli, A. e Tong, H., editors, *Computational Data and Social Networks*, p. 74–76, Cham. Springer International Publishing.

Optimization, G. (2019). Gurobi optimizer 8.1. *URL http://www. gurobi. com*.

Ping Wu, Ji-Rong Wen, Huan Liu, e Wei-Ying Ma (2006). Query selection techniques for efficient crawling of structured web sources. In *22nd International Conference on Data Engineering (ICDE'06)*, p. 47–47.

Qian, C., Yu, Y., e Zhou, Z.-H. (2015). Subset selection by pareto optimization. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., e Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, p. 1774–1782. Curran Associates, Inc. URL http://papers.nips.cc/paper/5822-subset-selection-by-pareto-optimization.pdf.

Romania, Q. S. (2010). Ant colony optimization applied to minimum weight dominating set problem. In *Proceedings of the 12th WSEAS International Conference on Automatic Control, Modelling & Simulation, Catania, Italy*, p. 29–31.

Shen, C. e Li, T. (2010). Multi-document summarization via the minimum dominating set. In *COLING*.

Wang, Y., Cai, S., Chen, J., e Yin, M. (2018). A fast local search algorithm for minimum weight dominating set problem on massive graphs. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, p. 1514–1522. AAAI Press. ISBN 9780999241127.

Wang, Y., Cai, S., e Yin, M. (2017). Local search for minimum weight dominating set with two-level configuration checking and frequency based scoring function. *Journal of Artificial Intelligence Research*, 58:267–295. ISSN 1076-9757. URL http://dx.doi.org/10.1613/jair.5205.

Zhu, X., Wang, W., Shan, S., Wang, Z., e Wu, W. (2012). A ptas for the minimum weighted dominating set problem with smooth weights on unit disk graphs. *Journal of Combinatorial Optimization*, 23:443–450.

Galoá { Este trabalho foi publicado utilizando o **Galoá** proceedings