A Biased Random Key Genetic Algorithm for the Weighted Independent Domination Problem

Guillem Rodríguez Corominas*
Universitat Politècnica de Catalunya
Barcelona, Spain
guillem.rodriguez.corominas@est.fib.upc.edu

Christian Blum
Artificial Intelligence Research Institute (IIIA-CSIC)
Bellaterra, Spain
christian.blum@iiia.csic.es

Maria J. Blesa
Universitat Politècnica de Catalunya
Barcelona, Catalonia
mjblesa@cs.upc.edu

ABSTRACT

This work deals with an *NP*-hard problem in graphs known as the weighted independent domination problem. We propose a biased random key genetic algorithm for solving this problem. The most important part of the proposed algorithm is a decoder that translates any vector of real-values into valid solutions to the tackled problem. The experimental results, in comparison to a state-of-the-art population-based iterated greedy algorithm from the literature, show that our proposed approach has advantages over the state-of-the-art algorithm in the context of the more dense graphs in which edges have higher weights than vertices.

CCS CONCEPTS

• **Theory of computation → Discrete optimization**;

KEYWORDS

biased random key genetic algorithm; weighted independent domination

ACM Reference Format:
Guillem Rodríguez Corominas, Christian Blum, and Maria J. Blesa. 2019. A Biased Random Key Genetic Algorithm for the Weighted Independent Domination Problem. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3319619.3326901

1 INTRODUCTION

The weighted independent domination (WID) problem is an *NP*-hard combinatorial optimization problem. The WID problem was initially introduced in [2]. Three integer linear programming (ILP) models and a population-based iterated greedy (PBIG) algorithm were presented in [5, 6]. In this work we make use of one of the two solution construction heuristics used by the PBIG algorithm in order to define an intelligent decoder for a so-called biased random key genetic algorithm (BRKGA). This type of GA was introduced in [3] and, since then, BRKGAs have been shown to obtain excellent results for a whole range of combinatorial problems including the maximum quasi-clique problem [7] and the project scheduling problem with flexible resources [1], to name a few of the more recent applications.

1.1 Preliminaries from Graph Theory

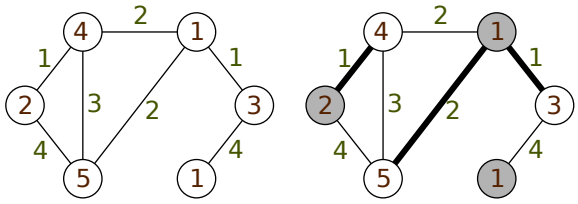
Before providing a technical description of the problem it is necessary to shortly introduce the required concepts from graph theory. An undirected graph is denoted by $G = (V, E)$, where V is a set of vertices and E the set of undirected edges. Hereby, an edge $e = (v_i, v_j) \in E$ connecting vertices $v_i, v_j \in V$ may equivalently be denoted by (v_j, v_i). Moreover, it is said that $e = (v_i, v_j)$ has two endpoints v_i and v_j, and thus $e \in E$ is said to be *incident* to them. When referring to the neighborhood of a vertex $v_i \in V$, we may refer to the *open neighborhood* $N(v_i) := \{v_j \in V \mid (v_i, v_j) \in E\}$ or to the *closed neighborhood* $N[v_i] := N(v_i) \cup \{v_i\}$. Given a vertex $v_i \in V$, another vertex $v_j \in V$ with $v_j \neq v_i$ and $v_j \in N(v_i)$ is called a *neighbor* of v_i in G. Furthermore, the set of edges connecting a vertex $v_i \in V$ to its neighbors in $N(v_i)$ is denoted by $\delta(v_i)$.

A *dominating set* in an undirected graph $G = (V, E)$ is a subset $D \subseteq V$ such that each vertex $v_i \in V$ is either part of D —that is, $v_i \in D$— or has at least one neighbor $v_j \in D$. Furthermore, an *independent set* in an undirected graph $G = (V, E)$ is a subset $I \subseteq V$ such that no two vertices $v_i \in I$ and $v_j \in I$ are connected by an edge from E. An independent set I of G is called maximal, if by adding any further vertex $v_i \in V \setminus I$ to I, set I would lose its property of being an independent set. It is important to note that any maximal independent set is a dominating set. Therefore, a maximal independent set is sometimes called an *independent dominating set*. To conclude these preliminaries, $N_D(v_i)$ denotes the *D-restricted neighborhood* of a vertex $v_i \in V \setminus D$ with relation to a given independent dominating set $D \subseteq V$, which is defined as follows: $N_D(v_i) := N(v_i) \cap D$, that is, the (open) neighborhood of v_i is restricted to all neighbors of v_i that are in D.

1.2 The Weighted Independent Domination Problem

The weighted independent domination (WID) problem can formally be described as follows. The input to the problem is an undirected weighted graph $G = (V, E, w)$ with vertex and edge weights. More specifically, each $v_i \in V$, respectively each $e \in E$, has an integer weight $w(v_i) \geq 0$, respectively $w(e) \geq 0$. The WID problem asks for finding an independent dominating set D in G minimizing the

*Undergraduate Student



(a) Input graph. (b) Optimal solution.

Figure 1: Example of the WID problem. (a) shows a possible input graph in which vertices and edges are labelled with their weights, and (b) shows the corresponding optimal solution.

following objective function:

$$f(D) := \sum_{v_j \in D} w(v_j) + \sum_{v_i \in V \setminus D} \min\{w(v_i, v_j) \mid v_j \in N_D(v_i)\} \quad (1)$$

Expressed in words, the objective function value of D is calculated as the sum of the weights of the vertices in D in addition to the sum of the weights of the lowest-weight edges that connect those vertices not in D to vertices in D. An example for the WID problem is shown in Figure 1. Both vertices and edges are labelled with their respective weights. Figure 1a shows a possible input graph G. The optimal solution to this input graph is shown in Figure 1b. The vertices forming part of the optimal solution are marked with a gray background. The lowest-weight edges that connect vertices not in D to nodes in D are indicated in bold. The objective function value of the optimal solution is 8, which is composed of the vertex weights (2 + 1 + 1) and the edge weights (2 + 1 + 1).

2 A BRKGA FOR THE WID PROBLEM

A BRKGA is a steady-state genetic algorithm which consists of a problem-independent part and a problem-dependent part. In the following we first describe the problem-independent part, whose pseudo-code is provided in Algorithm 1. The algorithm starts by calling function GenerateInitialPopulation(p_{size}), which generates a population P composed by p_{size} random individuals. Each individual $\pi \in P$ is a vector of length $|V|$ (where V is the set of vertices from the input graph) and the value at position i of π (denoted by $\pi(i)$) is a random real value from $[0, 1]$. Note that value $\pi(i)$ is associated to vertex v_i of the input graph G. Next, the individuals from the initial population are evaluated, which is the problem-dependent part of the algorithm described in the following subsection. After this evaluation, each individual $\pi \in P$ has an objective function value denoted by $f(\pi)$. Then, at each iteration of the algorithm, the following actions are performed. First, the best $\max\{\lfloor p_e \cdot p_{\text{size}} \rfloor, 1\}$ individuals are copied from P to P_e in function EliteSolutions(P, p_e). Second, a set of $\max\{\lfloor p_m \cdot p_{\text{size}} \rfloor, 1\}$ so-called mutants are generated and stored in P_m. These mutants are random individuals generated in the same way as the individuals from the initial population. Next, a set of $p_{\text{size}} - |P_e| - |P_m|$ individuals are generated by crossover in function Crossover($P, P_e, prob_{\text{elite}}$) and stored in P_c. Each such individual is generated as follows: (1) an elite parent π_1 is chosen uniformly at random from P_e, (2) a second parent π_2 is chosen uniformly at random from $P \setminus P_e$, and (3) an offspring individual

Algorithm 1 BRKGA for the WID Problem

```
1:  input: an undirected weighted graph G = (V, E, w)
2:  input: parameter values for p_size, p_e, p_m and prob_elite
3:  P := GenerateInitialPopulation(p_size)
4:  Evaluate(P)
5:  while computation time limit not reached do
6:      P_e := EliteSolutions(P, p_e)
7:      P_m := Mutants(P, p_m)
8:      P_c := Crossover(P, P_e, prob_elite)
9:      Evaluate(P_m ∪ P_c)          {NOTE: P_e is already evaluated}
10:     P := P_e ∪ P_m ∪ P_c
11: end while
12: output: Best solution in P
```

π_{off} is generated on the basis of π_1 and π_2 and stored in P_c. In the context of the crossover operator, value $\pi_{\text{off}}(i)$ is set to $\pi_1(i)$ with probability $prob_{\text{elite}}$, and to $\pi_2(i)$ otherwise. After generating all new offspring in P_m and P_c, these new individuals are evaluated in function Evaluate($P_m \cup P_c$). Note that the individuals in P_e are already evaluated. Finally, the population of the next generation is determined to be the union of P_e with P_m and P_c.

2.1 Evaluation of an Individual

The evaluation of an individual (see lines 4 and 9 of Algorithm 1) is the problem-dependent part of our algorithm. In the context of a BRKGA, the function that evaluates an individual is generally called the *decoder*. In our implementation of a BRKGA for the WID problem, the decoding of an individual π involves the application of two greedy heuristics that were introduced in [6]. These two greedy heuristics are described in the following.

Both greedy heuristics (henceforth denoted by GREEDY1 and GREEDY2) generate a solution in terms of an independent domination set $S \subset V$ as follows. They start with an empty solution $S = \emptyset$ and add, at each step, exactly one vertex from the so-called *remaining* graph $G' = (V', E')$ to S. They differ in the greedy function that is employed for choosing this vertex. G' is initially a copy of G. After adding a vertex $v \in V'$ to S, all vertices from $N[v \mid G']$ — that is, the closed neighborhood of v in G' — are removed from V'. In addition, all their incident edges are removed from E'. In this way, the only vertices that can be added to S are those that ensure that S maintains the property of being an independent set. In case V' is empty, S is a complete solution, and the algorithm stops. The pseudo-code of this procedure is shown in Algorithm 2.

Greedy function η_1 of GREEDY1. This greedy function is defined as follows:

$$\eta_1(v_i) := \frac{|N(v_i \mid G')| + 1}{w(v_i)} \quad , \text{ for all } v_i \in V'$$

where $N(v_i \mid G')$ refers to the neighborhood of v_i in G'. The vertex with the highest η_1-value is chosen. In other words, vertices with a high degree in the remaining graph G' and with a low vertex weight are preferred.

Greedy function η_2 of GREEDY2. For the description of η_2, the following notations are required. First, let $w_{\max} := \max\{w(e) \mid e \in E\}$. Second, we make use of an *auxiliary objective function*

Algorithm 2 Pseudo-code of Greedy1 and Greedy2

input: an undirected weighted graph $G = (V, E, w)$
$S := \emptyset$
$G' := G$
while $V' \neq \emptyset$ **do**
  Choose $v^* \in V'$ by using greedy function η_1 (for Greedy1) or greedy function η_2 (for Greedy2).
  $S := S \cup \{v^*\}$
  Remove from G' all nodes from $N[v^* \mid G']$ and their incident edges
end while
output: An independent dominating set S of G

$f^{\text{aux}}(.)$ for the evaluation of partial solutions. More precisely, given a partial solution S, $f^{\text{aux}}(S) := \sum_{v_i \in V} c(v_i \mid S)$, where

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

Given this definition of an auxiliary objective function, greedy function η_2 is defined as follows:

$$\eta_2(v_i) := f^{\text{aux}}(S) - f^{\text{aux}}(S \cup \{v_i\}) \quad , \text{ for all } v_i \in V'$$

The vertex with the highest η_2-value is chosen.

Greedy1 can then be used in the following way for the evaluation of an individual π. First, greedy function η_1 is replaced by the following greedy function:

$$\eta_1'(v_i) := \frac{|N(v_i \mid G')| + 1}{w(v_i)} \cdot \pi(i) \quad , \text{ for all } v \in V'$$

In other words, the values of the individual are used in order to bias the original greedy function weights. Then, Greedy1 is applied using the η_1' function. Let us denote the objective function value of the solution produced by Greedy1 on the basis of an individual π as $f_1(\pi)$.

A very similar mechanism is used for evaluating individual π by means of Greedy2. However, in this case we are faced with a potential problem, because η_2 may actually produce negative values for certain vertices. Therefore, when given a partial solution S, we first calculate

$$\eta_2^{\min} := \min\{\eta_2(v_i) \mid v_i \in V'\}$$

Then, η_2 is replaced by the following greedy function:

$$\eta_2'(v_i) := (\eta_2(v_i) - \eta_2^{\min} + 1) \cdot \pi(i) \quad , \text{ for all } v_i \in V'$$

Let us denote the objective function value of the solution produced by Greedy2 on the basis of an individual π as $f_2(\pi)$. The evaluation function of our BRKGA algorithm finally assigns value $\min\{f_1(\pi), f_2(\pi)\}$ to an individual π.

3 EXPERIMENTAL EVALUATION

The following algorithms were chosen for the comparison: (1) The two greedy approaches Greedy1 and Greedy2, (2) the population-based iterated greedy (Pbig) algorithm from [6], and (3) our BRKGA approach (henceforth labelled Brkga). Note that in [6] it was shown that when Pbig is applied within a framework known as Construct, Merge, Solve & Adapt (CMSA), it works even slightly better than when running it in a standalone fashion. However, we decided against the comparison of Brkga to this latter approach, because it is the goal of this preliminary work to develop a standalone approach that works better than the standalone version of Pbig, with the future aim of applying Brkga within the CMSA framework. All experiments concerning Brkga were run on a machine with an Intel(R) Core(TM) i7-6900K processor with 3.2 GHz, and a computation time limit of 300 seconds per input graph was used (just like in the case of Pbig).

Problem instances. For the experimental evaluation we chose a subset of the problem instances introduced in [6]. In particular, we chose all graphs with 100 vertices. They are classified according to type (random graphs vs. random geometric graphs), according to their density (three different density levels), and according to their vertex weight scheme (N: neutral; V: vertex-oriented, that is, vertices have higher weights than edges; E: edge-oriented, that is, edges have higher weights than vertices). The instance set consists of 10 graphs for each combination of these three factors. For more information we refer the interested reader to [6].

Tuning of Brkga. Our Brkga approach has four parameter for which well-working values must be found. For this purpose we used the automatic configuration tool irace [4]. In fact, we applied two different tuning runs: one for random graphs and the other one for random geometric graphs. For each combination of the graph density and the vertex weight scheme we selected the first one of the 10 available problem instances for tuning. This results in a total of nine tuning instances for each run. Moreover, the following parameter value ranges were considered:

- $p_{\text{size}} \in \{10, 20, 50, 100, 200, 500\}$.
- $p_e \in \{0.05, 0.1, 0.15, 0.2, 0.25\}$.
- $p_m \in \{0.1, 0.15, 0.2, 0.25, 0.3\}$.
- $prob_{\text{elite}} \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$.

In total, we allowed a maximum of 2000 experiments for each tuning run. The results provided by irace were as follows.

- Random graphs: $p_{\text{size}} = 500$, $p_e = 0.25$, $p_m = 0.2$, $prob_{\text{elite}} = 0.6$.
- Random geometric graphs: $p_{\text{size}} = 20$, $p_e = 0.2$, $p_m = 0.2$, $prob_{\text{elite}} = 0.7$.

These parameter value settings were used for the final experimentation. Finally, note that the tuning of Pbig (on the same set of problem instances) is described in [6].

Numerical results. Both Pbig and Brkga were applied—with a computation time limit of 300 seconds—exactly once to each problem instance. The numerical results (averaged over 10 problem instances per table row) are provided in Table 1 (random graphs) and in Table 2 (random geometric graphs). The first table column provides the weighting scheme and the second column the value of the parameter that indicates the density of the graphs (the edge probability *ep* in case of random graphs, and the radius r in the case of random geometric graphs). Note that the best result per row is shown with a light-gray background. The results allow to make the following observations:

Table 1: Numerical results for random graphs.

Weight scheme	*ep*	Greedy1		Greedy2		Pbig		Brkga	
		result	**time**	**result**	**time**	**result**	**time**	**result**	**time**
N	0.05	3589.1	<0.1	3519.1	<0.1	3049.8	0.54	3049.8	29.3
	0.15	3014.4	<0.1	2981.3	<0.1	2330.9	28.31	2330.9	30.6
	0.25	2883.5	<0.1	2796.1	<0.1	2070.9	0.16	2072.8	30.9
V	0.05	10465.6	<0.1	11756.6	<0.1	7747.0	76.47	7719.7	4.8
	0.15	4891.6	<0.1	5845.4	<0.1	3050.3	16.9	3046.6	28.8
	0.25	3297.5	<0.1	3488.9	<0.1	1808.4	3.5	1808.4	0.8
E	0.05	25698.7	<0.1	22269.3	<0.1	14378.7	0.78	14447.7	35.3
	0.15	27528.4	<0.1	23404.5	<0.1	14687.8	0.19	14632.8	77.8
	0.25	25451.4	<0.1	21770.0	<0.1	14506.6	<0.1s	14405.6	28.4

Table 2: Numerical results for random geometric graphs.

Weight scheme	*r*	Greedy1		Greedy2		Pbig		Brkga	
		result	**time**	**result**	**time**	**result**	**time**	**result**	**time**
N	0.14	3870.6	<0.1	3646.4	<0.1	3261.1	11.93	3261.1	5.6
	0.24	3798.8	<0.1	3378.1	<0.1	2882.5	3.04	2882.5	2.6
	0.34	3766.6	<0.1	3388.1	<0.1	2828.0	0.7	2833.5	19.3
V	0.14	7364.9	<0.1	7514.3	<0.1	5731.8	12.43	5731.8	7.9
	0.24	2880.1	<0.1	2724.2	<0.1	1981.8	<0.1s	1981.8	0.2
	0.34	1741.0	<0.1	1832.3	<0.1	968.8	<0.1s	940.5	0.1
E	0.14	29011.6	<0.1	24998.3	<0.1	19313.2	117.61	19190.0	3.4
	0.24	35312.1	<0.1	28647.1	<0.1	22108.3	6.36	22065.9	94.1
	0.34	37929.4	<0.1	30503.4	<0.1	23900.0	1.14	23754.8	8.2

- Both Pbig and Brkga clearly outperform the two greedy algorithms.
- For graphs of weight schemes N (neutral) and V (vertex-oriented) the performance of Pbig and Brkga is comparable.
- Brgka has an advantage over Pbig in the context of edge-oriented graphs (weight scheme E) when the graphs become more dense.
- In general, the computation time of Pbig for the densest graphs is very low, which does not happen for Brkga. This indicates that Pbig, in contrast to Brkga, gets stuck rather easily in local minima for these graphs.

4 CONCLUSIONS AND OUTLOOK

In this work we have proposed a biased random key genetic algorithm for solving the weighted independent domination problem in undirected graphs. The proposed decoder that translates vectors of real values into valid solutions to the tackled problem is based on learning the weights of two greedy algorithms. The results show that our algorithm has advantages over the current state-of-the-art algorithm for what concerns the more dense graphs produced under a weight scheme in which edges have higher weights than vertices.

In future work we will extend the experimental evaluation to larger graphs (the related literature provides, for example, graphs of 500 and 1000 vertices). Moreover, we will intent to improve the decoder extending by the employed greedy algorithms with a rollout mechanism.

It is the goal of this preliminary work to apply Brkga within a Construct, Merge, Solve & Adapt (CMSA) framework, and to compare it with the hybrid algorithm proposed in [6], which applies Pbig within a CMSA framework.

ACKNOWLEDGMENTS

Maria J. Blesa acknowledges support by MINECO and FEDER funds under grant GRAMM (ref. TIN2017-86727-C2-1-R) and by the Agency for Management of University and Research Grants (AGAUR) of the Government of Catalonia under project 2017 SGR 786 (ALBCOM).

REFERENCES

[1] B. F. Almeida, I. Correia, and F. Saldanha-da Gama. 2018. A biased random-key genetic algorithm for the project scheduling problem with flexible resources. *Top* 26, 2 (2018), 283–308.

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

[3] J. F. Gonçalves and M. G. C. Resende. 2011. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics* 17, 5 (2011), 487–525.

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

[5] P. Pinacho Davidson, C. Blum, and J. A. Lozano. 2017. The Weighted Independent Domination Problem: ILP Model and Algorithmic Approaches. In *Proceedings of EvoCOP 2017 – 17th European Conference on Evolutionary Computation in Combinatorial Optimisation (Lecture Notes in Computer Science)*, B. Hu and M. López-Ibáñez (Eds.), Vol. 10197. Springer Verlag, Berlin, Germany, 201–214.

[6] P. Pinacho Davidson, C. Blum, and J. A. Lozano. 2018. The weighted independent domination problem: Integer linear programming models and metaheuristic approaches. *European Journal of Operational Research* 265, 3 (2018), 860–871.

[7] B. Q. Pinto, C. C. Ribeiro, I. Rosseti, and A. Plastino. 2018. A biased random-key genetic algorithm for the maximum quasi-clique problem. *European Journal of Operational Research* 271, 3 (2018), 849–865.