Superior Genetic Algorithms for the Target Set Selection Problem Based on Power-Law Parameter Choices and Simple Greedy Heuristics

Benjamin Doerr
Laboratoire d'Informatique (LIX), CNRS, École Polytechnique, Institut Polytechnique de Paris
Palaiseau, France

Martin S. Krejca
Laboratoire d'Informatique (LIX), CNRS, École Polytechnique, Institut Polytechnique de Paris
Palaiseau, France

Nguyen Vu
École Polytechnique, Institut Polytechnique de Paris
Palaiseau, France

ABSTRACT

The target set selection problem (TSS) asks for a set of vertices such that an influence spreading process started in these vertices reaches the whole graph. The current state of the art for this NP-hard problem are three recently proposed randomized search heuristics, namely a biased random-key genetic algorithm (BRKGA) obtained from extensive parameter tuning, a max-min ant system (MMAS), and a MMAS using Q-learning with a graph convolutional network.

We show that the BRKGA with two simple modifications and without the costly parameter tuning obtains significantly better results. Our first modification is to simply choose all parameters of the BRKGA in each iteration randomly from a power-law distribution. The resulting parameterless BRKGA is already competitive with the tuned BRKGA, as our experiments on the previously used benchmarks show.

We then add a natural greedy heuristic, namely to repeatedly discard small-degree vertices that are not necessary for reaching the whole graph. The resulting algorithm consistently outperforms all of the state-of-the-art algorithms.

Besides providing a superior algorithm for the TSS problem, this work shows that randomized parameter choices and elementary greedy heuristics can give better results than complex algorithms and costly parameter tuning.

CCS CONCEPTS

• **Mathematics of computing → Combinatorial optimization**.

KEYWORDS

Target set selection, combinatorial optimization, biased random-key genetic algorithm, parameter tuning

ACM Reference Format:

Benjamin Doerr, Martin S. Krejca, and Nguyen Vu. 2024. Superior Genetic Algorithms for the Target Set Selection Problem Based on Power-Law Parameter Choices and Simple Greedy Heuristics. In *Genetic and Evolutionary Computation Conference (GECCO '24), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3638529.3654140

1 INTRODUCTION

The *target set selection* problem [18] (TSS) is a combinatorial optimization problem that aims to find in a given graph a smallest subset of vertices (the *target set*) such that a diffusion process started on this set eventually reaches all vertices. Finding such a target set has relevant applications in various domains, especially in viral marketing [17, 22]. However, as the TSS problem is NP-hard [18] and not well approximable [5], it is typically solved heuristically. The state-of-the-art heuristics are the randomized search heuristics that have been proposed in two recent papers [20, 24].

The first of these works, by López Serrano and Blum [20], develops a biased random-key genetic algorithm (BRKGA), which encodes potential solutions as a vector of floating-point numbers between 0 and 1. Each such vector is translated via a greedy heuristic into a valid target set, that is, a set such that the diffusion process on the instance graph actually reaches all vertices. The authors tune the parameters of the BRKGA and compare it to the previous state-of-the-art heuristic (called *minimum target set* (MTS) [6]). On hard instances, they find that the BRKGA almost always finds solutions that are at least 10 % better than those of the MTS, and this in significantly shorter time, especially for large instances.

The second work, by Ramírez Sánchez et al. [24], considers two versions of a max-min ant system (MMAS). One version is a classic MMAS (called MMAS), the other version uses information obtained from a Q-learning approach combined with a graph convolutional network (called MMAS-Learn). The authors tune the parameters of both versions and then compare them with the BRKGA above [20]. They observe that the two MMAS variants find better solutions than the BRKGA on most of the hard instances. Especially, the MMAS-Learn is best on large networks.

All of these algorithms have in common that they require considerable effort up front in order to be used to their full effectiveness. The BRKGA [20] has four parameters with a wide range of possible values. The MMAS variants [24] have three parameters each. Each approach is tuned by the authors via *irace* [21] before conducting the experiments. This tuning is computationally very expensive. For example, both López Serrano and Blum [20] and Ramírez Sánchez et al. [24] report that they used irace with a budget of 2 000 algorithm runs in order to tune the parameters. In comparison, when conducting the experimental evaluation, in each of the two papers, the authors perform 10 runs for each of the 27 instances. The Q-learning approach combined with the graph convolutional network used for one MMAS variant [24] is already a very complex and costly process in itself, for example, requiring to generate training instances for the graph convolutional network. Overall, since López Serrano and Blum [20] compared their BRKGA to an existing state-of-the-art TSS heuristic and achieved clearly superior results, this begs the question whether more complex and costly heuristics are necessary in order to achieve good results for the hard TSS problem.

Our contribution. We show that the TSS problem can be solved heuristically without the need for complex parameter tuning or other heavy machinery, such as graph convolutional networks. We propose two modifications to the BRKGA by López Serrano and Blum [20], in particular, removing the need for parameter tuning. Our resulting algorithm outperforms the state-of-the-art algorithms by López Serrano and Blum [20] and Ramírez Sánchez et al. [24] on almost all hard instances with statistical significance in terms of solution quality (Table 6). In terms of computation time, our approach is comparable, while not requiring any offline computations, such as parameter tuning.

Our first modification is to replace the costly parameter tuning by a simple randomized parameter choice in each iteration. For this randomized choice, we use a power-law distribution, cropped and scaled so that it covers exactly the range of parameter values used in [20] for the *irace* parameter tuning.

The idea for this type of parameter definition stems from the theoretical work [11], where this approach was used for the first time in discrete evolutionary computation, namely to set the mutation rate of a simple evolutionary algorithm. As shown in [11], this parameter choice gave a drastic speed-up compared to the standard mutation rate $1/n$, and it was only slightly inferior to the instance-dependent optimal mutation rate. The idea to choose parameters in this randomized fashion was quickly taken up in other theoretical works, including works where the power-law choice gave a performance asymptotically faster than with any fixed parameter value [2] or where several parameters where chosen in this fashion simultaneously [3].

Despite this success in theoretical works, power-law distributed parameter choices have not really made it into practical applications of evolutionary algorithms. Our empirical evaluation shows that there is no reason for this. By simply replacing the parameter choices obtained from expensive tuning by power-law distributed parameters, we obtain a variant of the BRKGA of [20] that has a comparable performance (but needed no tuning in the design).

Our second modification is a simple TSS-specific local heuristic that aims at reducing the size of already valid target sets. It greedily eliminates vertices in the order of increasing vertex degree, always checking whether the result is still valid after the removal. In addition to adding this modification to the BRKGA, we also combine it with another easy TSS-specific heuristic by López Serrano and Blum [20]. In our empirical evaluation with the other approaches, this combination even achieves the best results on some hard instances.

Overall, we observe that easy adjustments can drastically improve the quality for TSS heuristics. While our experiments are specific to the TSS problem, we believe that the insights carry over to other problems as well. Coming up with an easy problem-specific heuristics for improving solutions locally is a good choice in general and essentially the same as finding reduction rules, which are very popular, for example, in the PACE challenge [1]. Furthermore, our recommendation for choosing parameter values on the fly via a power-law distribution is not limited to the TSS problem or the BRKGA, and we invite researchers to try it themselves.

2 THE TARGET SET SELECTION PROBLEM

The *target set selection* problem (TSS [18]) is an NP-hard graph optimization problem. At its core is a discrete-time diffusion process that determines how vertices in a graph become *active*. The TSS problem aims to find a minimum cardinality set of initially active vertices *(target set)* such that the diffusion process eventually activates all vertices in the graph.

To make this more precise, for an undirected graph $G = (V, E)$ and for all $v \in V$, let $\Gamma(v) = \{u \in V \mid \{u, v\} \in E\}$ denote the open neighborhood of v, and let $\deg(v) = |\Gamma(v)|$ denote the degree of v. Given an undirected graph $G = (V, E)$, a threshold function $\theta\colon V \to \mathbb{N}$ such that for all $v \in V$ it holds that $\theta(v) \leq \deg(v)$, as well as a subset $S \subseteq V$ of initially active vertices, the *diffusion process of G with θ and S* is defined as a sequence $(F_t)_{t\in\mathbb{N}}$ over subsets of vertices such that in each iteration vertices whose number of active neighbors is at least the threshold become active as well. Formally,

(1) $F_0 = S$ and
(2) $F_{t+1} = F_t \cup \{v \in V \mid |\Gamma(v) \cap F_t| \geq \theta(v)\}$ for all $t \in \mathbb{N}$.

Let $T = \min\{t \in \mathbb{N} \mid F_{t+1} = F_t\}$. Note that since the diffusion process is deterministic, the set of active vertices remains the same for all $t \geq T$. Since, by definition, the set of active vertices is strictly increasing for all iterations before T, the diffusion process stops after at most $|V| - 1$ iterations. Let $\sigma_\theta(S) := F_T$ denote the *finally active set* when starting with set S. We say that S is *valid* if and only if $\sigma_\theta(S) = V$.

Given an undirected graph $G = (V, E)$ and a threshold function θ, the TSS problem aims to find a minimum cardinality set $S^* \subseteq V$ such that $\sigma_\theta(S^*)$ is valid.

2.1 Objective Values for the TSS Problem

In order to solve the TSS problem heuristically, we need to assign an objective value (the *fitness*) to each solution candidate to the problem. Given a TSS instance $((V, E), \theta)$, each $S \subseteq V$ is a solution candidate. The fitness of S is $|S|$ if S is valid, and the fitness is $|V| + 1$ otherwise. Hence, the solution candidates with the smallest fitness are solutions to the original TSS instance. We note that we only consider algorithms in this article that consider valid solution candidates. Hence, technically, the fitness of invalid candidates is not required for our purposes.

3 EXISTING HEURISTICS FOR THE TARGET SET SELECTION PROBLEM

We now present, to the best of our knowledge, the most recent and best performing heuristics for the TSS problem, by López Serrano and Blum [20] and Ramírez Sánchez et al. [24]. López Serrano and Blum [20, Algorithm 1] propose a variant of the *biased random-key*

Algorithm 1: The maximum-degree heuristic (MDG [20, Algorithm 3]) that greedily constructs a valid target set.

```
Input: graph $G = (V, E)$, threshold function $\theta\colon V \to \mathbb{N}$
1 $S \leftarrow \emptyset$;
2 $\mathrm{Cov} \leftarrow \emptyset$;
3 while $\mathrm{Cov} \neq V$ do
4     $v^* \leftarrow \mathrm{argmax}_{v \in V \setminus \mathrm{Cov}} \deg(v)$;
5     $\mathrm{Cov} \leftarrow \sigma_\theta(\mathrm{Cov} \cup \{v^*\})$;
6     $S \leftarrow S \cup \{v^*\}$;
Output: $S$
```

genetic algorithm framework (BRKGA, Section 3.2), which is a genetic algorithm that uses mutation and crossover and that maintains a population of a fixed size. Ramírez Sánchez et al. [24] consider a *max-min ant system* algorithm (MMAS), with and without heuristic information retrieved via Q-learning from a graph convolutional neural network. The authors compare these MMAS variants empirically to the BRKGA by López Serrano and Blum [20] and find that the MMAS with Q-learning finds the best solutions among all three algorithms in 19 out of 27 cases (including ties), especially for all of the largest networks that were tested. However, also the MMAS variant without Q-learning as well as the BRKGA perform best in 11 and 10 cases, respectively (with ties).

All of these algorithms construct valid target sets greedily based via an adaptation of the *maximum-degree heuristic* (MDG) [20, Algorithm 3], see Section 3.1 below. López Serrano and Blum [20] compared their BRKGA to the MDG alone and observed that the BRKGA found strictly better solutions in all but 2 cases, which were ties.

3.1 Maximum-Degree Heuristic

The maximum-degree heuristic (MDG) [20, Algorithm 3], see also Algorithm 1, determines for a given graph $G = (V, E)$ and a threshold function θ a valid target set $S \subseteq V$, that is, a set such that the diffusion process of G with θ started in S eventually activates all vertices of G. To this end, MDG starts with the empty set and greedily adds a vertex of largest degree to its current solution. After each vertex added, it checks whether the set is already valid. Once it constructs a valid set, it returns it.

Note that in line 5, we compute the set $\sigma_\theta(S \cup \{v^*\})$ of vertices activated by $S \cup \{v^*\}$, exploiting that $\mathrm{Cov} = \sigma_\theta(S)$ and $\sigma_\theta(S \cup \{v^*\}) = \sigma_\theta(\sigma_\theta(S) \cup \{v^*\})$ [20, Proposition 1]. We perform such an update with a slightly modified breadth-first search.

3.2 Biased Random-Key Genetic Algorithm

The biased random-key genetic algorithm (BRKGA) [20, Algorithm 1], see also Algorithm 2, is an elitist genetic algorithm that maintains a population of given fixed size n_{ind} of *individuals.* For the TSS problem, each individual is represented as a vector of length $|V|$, with each component being a floating-point number in $[0, 1]$. The initial population contains only random individuals as well as one whose values are all 0.5. In each iteration, the BRKGA selects the best solutions from its population, resulting in the *elitist* population. Then, it creates a population of new random individuals and another population via crossover between the elitist population and the entire population from the previous iteration. The union of these two populations as well as the elitist population make up the population for the next iteration.

Algorithm 2: The biased random-key genetic algorithm (BRKGA [20, Algorithm 1]) for the target set selection problem. Each individual is a vector of length $|V|$, with components in $[0, 1]$.

```
Input: graph $G = (V, E)$, threshold function $\theta\colon V \to \mathbb{N}$
Input: parameter values $n_{\mathrm{ind}} \in \mathbb{N}_{\geq 1}$, $p_{\mathrm{e}}, p_{\mathrm{m}} \in [0, 1]$ with
        $p_{\mathrm{e}} + p_{\mathrm{m}} \leq 1$, $\mathrm{prob}_{\mathrm{elite}} \in [0, 1]$
1 $P \leftarrow$ population of $n_{\mathrm{ind}} - 1$ individuals, each generated uniformly at random;
2 $P \leftarrow P \cup \{(0.5)_{i \in V}\}$;
3 while termination criterion not met do
4     $P_{\mathrm{e}} \leftarrow$ the best $\lceil p_{\mathrm{e}} \cdot n_{\mathrm{ind}} \rceil$ individuals from $P$;
5     $P_{\mathrm{m}} \leftarrow$ population of $\lceil p_{\mathrm{m}} \cdot n_{\mathrm{ind}} \rceil$ individuals, each generated uniformly at random;
6     $P_c \leftarrow$ population of $n_{\mathrm{ind}} - |P_{\mathrm{e}}| - |P_{\mathrm{m}}|$ individuals created by crossover between $P$ and $P_{\mathrm{e}}$ with bias $\mathrm{prob}_{\mathrm{elite}}$;
7     $P \leftarrow P_e \cup P_m \cup P_c$;
Output: best solution in $P$
```

Decoding an individual. Following the approach by López Serrano and Blum [20], an individual $(w_i)_{i \in V}$ is decoded into a set $S \subseteq V$ that is valid for the TSS problem by following the same approach as MDG but guided by w instead of the vertex degrees. More specifically, the decoding performs Algorithm 1 but replaces line 4 by

$$v^* \leftarrow \mathrm{argmax}_{v \in V \setminus \mathrm{Cov}}\, w_v \cdot \deg(v);$$

Quality of an individual. The quality of an individual is determined by decoding it into a vertex set S and then determining the objective value of S as explained in Section 2.1. Note that the TSS problem is a minimization problem. Hence, the term *best* in Algorithm 2 refers to individuals with the smallest quality-value among the population (breaking ties arbitrarily).

Crossover. The crossover operation picks one individual $x \in P$ and one individual $y \in P_{\mathrm{e}}$, each uniformly at random. It then creates a new individual z by choosing for each component a value of one of the two parents, with a probability bias of $\mathrm{prob}_{\mathrm{elite}}$ toward y. That is, for all $v \in V$, it holds independently with probability $\mathrm{prob}_{\mathrm{elite}}$ that $z_v = y_v$, and it holds $z_v = x_v$ otherwise.

4 OUR IMPROVEMENTS TO THE EXISTING HEURISTICS

We propose two independent modifications to the BRKGA described in Section 3, namely:

(1) On-the-fly parameter choices based on a power-law distribution (Section 4.1).
(2) The *minimum-degree heuristic* (reverseMDG, Algorithm 3), which is given a valid solution candidate for the TSS problem and greedily aims to improve it.

4.1 On-the-Fly Parameter Tuning via Power-Law Random Choices

To avoid the costly tuning of the parameters of the BRKGA, we propose to choose the parameter values during the run, namely randomly from a power-law distribution. Naturally, this only concerns parameters that can be adjusted meaningfully during the run. For the BRKGA, these are all parameters except the population size, which we choose following the recommendation by López Serrano and Blum [20].

Power-law distributed random parameter values were proposed first (in evolutionary computation with discrete search spaces) in the theoretical work [11], where under the name *fast GA* it was suggested to use bit-wise mutation with a random mutation rate, sampled anew for each application of the mutation operator from a power-law distribution. This idea was quickly taken up in other theoretical works and extended to other parameters, to more than one parameter, and to multi-objective optimization [2–4, 7–9, 12–16, 23, 25]. All these work showed that a power-law parameter choice can significantly beat the standard parameter choice, or even any static parameter choice, and this in an essentially parameterless manner (formally speaking, the power-law exponent β is a parameter, but already the first work [11] detected that it has little influence on the performance and that taking $\beta = 1.5$ is a reasonable choice). Surprisingly, these promising theoretical results never found significant applications in practice.

Formal definition of capped power-laws. In [11], the following type of capped power-law was used. Let $\beta > 1$ be the (negative) power-law exponent and $[1..r] := [1, r] \cap \mathbb{N}$, $r \in \mathbb{N}$, be the range of the distribution. Then a random variable X follows a power-law with exponent β and range $[1..r]$ if for all $k \in [1..r]$,

$$\Pr[X = k] = \frac{k^{-\beta}}{C_r(\beta)},$$

where $C_r = \sum_{k=1}^{r} k^{-\beta}$.

Doerr et al. [11] report that the choice of β does not have a big impact on the run time, and they recommend a choice of $\beta = 1.5$. The range of the power-law is usually determined by the application. In [11], a random mutation rate for mutation of bit-strings of length n was sought, so with the aim of avoiding mutation rates larger than $1/2$, the number r was chosen as $r = \frac{n}{2}$, a number α was sampled from a power-law with range $[1..r]$, and α/n was used as mutation rate. These considerations turn the power-law distribution effectively into a parameter-less distribution.

It is clear that the range of the power-law distribution can be adjusted to arbitrary finite sets A by choosing an appropriate mapping between A and $[1..r]$.

4.1.1 *Applying Power-Law Tuning to the BRKGA.*

Out of the four parameters of the BRKGA (Algorithm 2), we choose all but n_{ind} via a power-law distribution.[1]

For the three remaining parameters, we pick a power-law over the range of values that López Serrano and Blum [20, Table 1] chose for their parameter tuning. We apply a suitable transformation, as reported in Table 1. We note that the ranges of p_e and p_m do not overlap and are such that the constraint $p_\text{e} + p_\text{m} \leq 1$ is always satisfied. However, we only make this choice because we choose for each parameter the same range as López Serrano and Blum [20, Table 1]. The constraint $p_\text{e} + p_\text{m} \leq 1$ can still be satisfied if the ranges of p_e and p_m would overlap. In such a case, one would first sample one of the two values, for example p_e, and then sample the other value via rejection sampling. Alternatively, the support of the power-law mutation for the second could be adjusted.

When transforming the values of the power-law into the intended parameter range, it is not very important where the highest probability mass lies, as the ranges are all rather small. For example, for p_e, the highest mass is on the value that López Serrano and Blum [20] determined as best via the parameter tuner irace [21]. However, for prob_elite, the best value reported by López Serrano and Blum [20] is 0.69, which receives a comparably low probability in the power-law we chose.

We recall that we draw a value for each of these three parameters once at the beginning of the while-loop of the BRKGA, that is, once each iteration.

[1]Technically, the BRKGA by López Serrano and Blum [20] has one more parameter, which is binary and affects a single individual in the initial population. Since it does not make sense to adjust such a parameter dynamically, we go with the choice that López Serrano and Blum [20] propose (i.e., seed = true). Algorithm 2 already reflects this choice.

Table 1: The power-law distributions we employ when choosing the parameters of the BRKGA (Algorithm 2) according to the method described in Section 4.1. The column *Support* denotes the values that the power-law distribution takes. All distributions choose a power-law exponent of 1.5. The conversion map shows how we transform the values from *Support* such that they result in a power-law over the values in *Resulting range*, which are the ones chosen by López Serrano and Blum [20, Table 1]. The value x represents a random number drawn from the respective power-law distribution.

Parameter	Support	Conversion map	Resulting range
p_e	[1..15]	$0.1 + 0.01(15 - x)$	$\{0.24, 0.23, \ldots, 0.1\}$
p_m	[1..20]	$0.1 + 0.01x$	$\{0.11, 0.12, \ldots, 0.3\}$
prob_elite	[1..30]	$0.5 + 0.01x$	$\{0.51, 0.52, \ldots, 0.8\}$

Algorithm 3: The minimum-degree heuristic (reverse-MDG) that is given a valid solution candidate for the target set selection problem and greedily tries to eliminate vertices of the lowest degree.

```
Input: graph $G$, threshold function $\theta$, and $S \subseteq V$ such that
       $\sigma_\theta(S) = V$
1 $V_\leq \leftarrow$ vertices in $V$ sorted by ascending vertex degree;
2 $C \leftarrow S$;
3 for $v \in V_\leq$ do
4     if $v \in C$ then
5         $C' \leftarrow C \setminus \{v\}$;
6         if $\sigma_\theta(C') = V$ then $C \leftarrow C'$;
Output: $C$
```

Table 2: The best TSS objective values achieved for all of the algorithms that we ran on the specified networks, as described in Section 5.1. The average is over 10 independent runs per algorithm. Bold numbers indicate the best value among all algorithms in this table and in Table 3. The columns $|V|$ and $|E|$ denote the number of vertices and edges, respectively, of the networks. See Section 5.2.3 for more details.

Network	$\|V\|$	$\|E\|$	BRKGA		MDG+rev	BRKGA+rev		fastBRKGA		fastBRKGA+rev	
			Best	Avg.		Best	Avg.	Best	Avg.	Best	Avg.
Dolphins	62	159	**6**	6.0	7	**6**	6.0	**6**	6.0	**6**	6.0
Football	115	613	**22**	23.4	28	**22**	23.4	**22**	23.5	**22**	23.5
Karate	34	78	**3**	3.0	**3**	**3**	3.0	**3**	3.0	**3**	3.0
Jazz	198	2 742	**20**	21.1	24	**20**	21.1	**20**	21.1	**20**	21.1
CA-AstroPh	18 772	198 050	1 428	1 438.0	1 381	**1 375**	1 384.9	1 427	1 438.4	**1 375**	1 385.6
CA-GrQc	5 242	14 484	928	930.7	**889**	891	895.7	924	930.6	892	897.0
CA-HepPh	12 008	118 489	1 343	1 347.8	**1 257**	1 280	1 286.0	1 338	1 349.2	1 278	1 285.7
CA-HepTh	9 877	25 973	1 234	1 242.4	**1 154**	1 155	1 160.2	1 237	1 242.4	1 157	1 160.6
CA-CondMat	23 133	93 439	2 563	2 592.8	**2 326**	2 350	2 360.4	2 580	2 599.5	2 354	2 364.1
Email-Enron	36 692	183 831	2 670	2 679.1	2 676	2 635	2 643.5	2 664	2 671.7	**2 633**	2 639.1
ego-facebook	4 039	88 234	466	470.5	477	463	464.7	467	473.2	**460**	466.7
socfb-Brandeis99	3 898	137 567	**320**	337.4	356	**320**	336.2	321	332.4	**320**	331.5
socfb-nips-ego	2 888	2 981	**10**	10.0	**10**	**10**	10.0	**10**	10.0	**10**	10.0
socfb-Mich67	3 748	81 903	166	168.1	177	165	168.0	**164**	167.9	**164**	167.9
soc-gplus	23 628	39 194	62	63.0	**61**	**61**	61.0	62	62.8	**61**	61.1
musae_git	37 700	289 003	173	178.4	190	172	178.1	172	178.3	**171**	178.1
loc-gowalla_edges	196 591	950 327	5 450	5 465.2	4 780	**4 760**	4 777.2	5 436	5 464.6	4762	4 778.9
gemsec_facebook_artist	50 515	819 090	648	669.2	688	631	649.8	624	671.1	**610**	650.6
deezer_HR	54 573	498 202	2 046	2 092.1	1 895	1 874	1 904.5	2 057	2 096.5	**1 873**	1 892.2
com-dblp	317 080	1 049 866	36 875	36 917.0	**29 114**	29 170	29 215.8	36 832	36 907.8	29 212	29 235.5
Amazon0302	262 111	899 792	35 579	35 567.3	**26 618**	26 665	26 642.0	35 576	35 563.9	26 667	26 636.4
Amazon0312	400 727	2 349 869	31 051	31 075.2	23 529	**23 523**	23 549.6	31 045	31 068.2	23 542	23 559.4
Amazon0505	410 236	2 439 437	31 763	31 778.5	**24 115**	**24 115**	24 134.6	31 751	31 776.9	24 122	24 135.0
Amazon0601	403 394	2 443 408	31 393	31 412.3	23 759	**23 757**	23 777.1	31 390	31 403.0	23 758	23 777.6

4.2 ReverseMDG

The minimum-degree heuristic (reverseMDG, Algorithm 3) is given a valid TSS solution candidate S and greedily tries to eliminate vertices from S, starting with those of the lowest degree. In our BRKGA, we apply the heuristic after obtaining target sets from the MDG heuristic (Algorithm 1).

Since the TSS problem is NP-hard, a heuristic solution candidate, such as one constructed by MDG, may not be optimal because (1) it may not contain vertices that are required for an optimal solution or (2) it may not have minimum size. While reverseMDG does not help with (1), it helps with (2). Since vertices with a high degree can potentially activate more vertices once they are active themselves, reverseMDG tries to remove vertices with a low degree first. Although this does not even guarantee a locally optimal solution, we see in Section 5 that reverseMDG is already very powerful. We note that, mostly due to its simplicity, reverseMDG is also comparably cheap to compute: It only needs to compute the finally active set for a given set at most $|S|$ times, which is typically not too large, as we are considering a minimization problem.

5 EMPIRICAL EVALUATION

We compare the existing heuristics for the TSS problem discussed in Section 3 to a select choice of these algorithms modified by our approaches discussed in Section 4. More specifically, we compare the BRKGA from López Serrano and Blum [20] as well as the MMAS with and without Q-learning from Ramírez Sánchez et al. [24] to the BRKGA with power-law tuning (Section 4.1.1), the BRKGA with power-law tuning and with reverseMDG, and the MDG with reverseMDG. We compare these algorithms primarily with respect to the best solution that they found after a fixed time budget. However, we also report on the run time. Our code is available on GitHub [10].

We chose to apply our modifications to the BRKGA and the MDG, because they are simple algorithms, and we are interested to see to what extent our modifications improve such simple approaches, especially compared to more sophisticated approaches such as MMAS with Q-learning. For modifications that employ the reverseMDG, we add the suffix *+rev* to the algorithm name. For modifications that employ the power-law parameter tuning, we add the prefix *fast* to the algorithm name, based on the naming convention by Doerr et al. [11].

5.1 Experimental Setup

We consider the same experimental setup as López Serrano and Blum [20] and Ramírez Sánchez et al. [24]. To this end, we consider 24 out of the 27 social networks that the previous studies considered that are part of the Stanford Network Analysis Project [19]. We note

Table 3: The empirical results from Ramírez Sánchez et al. [24, Table 2] for solving the TSS instances as explained in Section 5.1. The numbers are the objective values of the best solution found after a certain time budget. The average is over 10 independent runs per network. Numbers in bold are the best value among all of the algorithms in this table and in Table 2. We note that we strongly believe that Ramírez Sánchez et al. [24, Table 2] mixed up the labels of the rows for `Dolphins` and for `Karate`, as the number of vertices and edges do not match the actual values of these networks. Below, we corrected this. See Section 5.2.3 for more details.

Network	MMAS		MMAS-Learn	
	Best	Avg.	Best	Avg.
Dolphins	**6**	6.0	**6**	6.0
Football	23	23.0	**22**	23.0
Karate	**3**	3.0	**3**	3.0
Jazz	**20**	20.0	**20**	20.0
CA-AstroPh	1 405	1 412.5	1 405	1 413.0
CA-GrQc	898	900.1	897	899.4
CA-HepPh	1 289	1 297.2	1 289	1 298.4
CA-HepTh	1 179	1 186.2	1 182	1 189.2
CA-CondMat	2 416	2 422.3	2 419	2 428.0
Email-Enron	2 679	2 686.0	2 692	2 699.4
ego-facebook	478	482.8	478	481.9
socfb-Brandeis99	365	369.2	368	370.2
socfb-nips-ego	**10**	10.0	**10**	10.0
socfb-Mich67	177	179.3	177	179.3
soc-gplus	**61**	61.5	**61**	61.9
musae_git	202	205.9	199	206.4
loc-gowalla_edges	5 180	5 195.1	5 155	5 177.6
gemsec_facebook_artist	726	744.7	735	748.4
deezer_HR	2 231	2 247.0	2 240	2 255.9
com-dblp	32 981	33 016.9	32 364	32 397.7
Amazon0302	30 291	30 334.7	29 553	29 618.0
Amazon0312	26 280	26 317.8	26 186	26 201.2
Amazon0505	26 945	27 000.9	26 801	26 871.2
Amazon0601	26 665	26 708.3	26 511	26 568.0

that we exclude the network `com-youtube` from this data set, due to its size—being at least 2.5 times larger than the largest of the remaining networks, in terms of vertices. The two networks `ncstrlwg2` and `actors-data` from the previous studies are excluded because we could not obtain them.

For each graph (V, E) above, we consider a TSS instance where the threshold of each vertex is half its degree, that is, for each $v \in V$, it holds that $\theta(v) = \lceil \deg(v)/2 \rceil$.

We run the original BRKGA from López Serrano and Blum [20] with the best parameters determined by the authors. In addition, we run our modifications MDG+rev, BRKGA+rev, fastBRKGA, and fastBRKGA+rev. Since the BRKGA+rev does not tune its parameters automatically, we choose the best parameters determined by López Serrano and Blum [20]. For all versions of the BRKGA, we choose the population size $n_{\text{ind}} = 46$ recommended by López Serrano and Blum [20]. We execute 10 independent runs per algorithm and TSS

Table 4: The statistical significance of the best solution found of the *fastBRKGA compared to the (tuned) BRKGA* by López Serrano and Blum [20]. The reported p-values are the result of a Mann–Whitney U test based on the same 10 independent runs per algorithm per network as the ones in Table 2. Values of statistical significance (that is, a p-value of at most 0.05) are highlighted in bold. See Section 5.2.2 for more details.

Network	p-value
Dolphins	1.00
Football	0.77
Karate	1.00
Jazz	0.56
CA-AstroPh	0.82
CA-GrQc	0.91
CA-HepPh	0.70
CA-HepTh	0.79
CA-CondMat	0.57
Email-Enron	**0.03**
ego-facebook	0.10
socfb-Brandeis99	0.27
socfb-nips-ego	1.00
socfb-Mich67	0.97
soc-gplus	0.36
musae_git	1.00
loc-gowalla_edges	1.00
gemsec_facebook_artist	0.60
deezer_HR	0.60
com-dblp	0.62
Amazon0302	0.33
Amazon0312	0.24
Amazon0505	1.00
Amazon0601	**0.05**

instance. For each graph (V, E), each run has a run time budget of $\max\{100, |V|/100\}$ seconds. We stop a run prematurely if it finds an optimal solution. During each run, we log the objective value of the best solution found so far. For the BRKGA and the fastBRKGA, we also log the run times.

All of our modifications were implemented in C++, since López Serrano and Blum [20] also used C++ for the BRKGA. Our experiments were run on a machine with 2 Intel® Xeon® Platinum 8362 CPUs @ 2.80 GHz (32 cores; 64 threads) with 2 TB of RAM.

5.2 Results

We discuss our results mostly with respect to the quality of the best solution found. We do so first in absolute terms (Section 5.2.1) and then in terms of statistical significance (Section 5.2.2). Last, we briefly discuss the run time, since we compare our results to some algorithms that we did not run ourselves (Section 5.2.3).

5.2.1 Comparison of the Best Objective Value. Table 2 shows our results in terms of objective value. Table 3 shows the results obtained by Ramírez Sánchez et al. [24] for the same setting; that is, we did not run experiments for the results in Table 3.

First, we observe that all bold entries for the existing heuristics (that is, BRKGA, MMAS, and MMAS-learn) are almost always matched by all of our modifications of the BRKGA. The only two exceptions are that the fastBRKGA reports for `soc-gplus` and for `socfb-Brandeis99` values that are worse by 1 than the overall best value. For all of these networks, the average is for all algorithms also very close to the best value, indicating that these are not particularly hard instances. Still, our modification MDG+rev does not match the best value in 3 cases, showing that a purely greedy strategy is insufficient. In total, these cases cover 6 out of the 24 networks.

For the remaining 18 networks on which at least one of our modifications is solely the best algorithm, we get a diverse picture. Surprisingly, the deterministic greedy heuristic MDG+rev obtains the best value in 7 of these 18 cases. This implies for these instances that the randomness used by any of the other algorithms is worse than a simple one-time greedy approach. This is even the case for the fastBRKGA+rev, which employs a modified version of MDG+rev in order to create valid solutions. This suggests that it would be generally better to add the solution computed by MDG+rev to the initial population of every other algorithm, which would remain in the population, due to the elitist nature of the algorithms.

Focusing on our modifications of the BRKGA on the $18 - 7 = 11$ remaining networks, we see that the fastBRKGA performs worst in terms of number of best solutions found. However, when compared to the original BRKGA, which was tuned in advance, the results are very similar. This shows that the power-law parameter tuning is actually comparable to the offline parameter tuning of the BRKGA via irace. We consider this to be a success for the power-law tuning.

The two remaining algorithms are the BRKGA+rev and the fastBRKGA+rev, which only differ on an algorithmic level with respect to whether their parameters were tuned before the optimization or chosen during the optimization. We observe that the fastBRKGA+rev performs better on smaller networks, whereas the BRKGA+rev performs better on larger networks. However, for each of these networks, the differences in the best solution found are rather close. Any of these two modified algorithms is better than the state of the art, presented in Table 3. Overall, this shows that the reverseMDG heuristic, which is problem-dependent, has an important impact on the overall performance. Furthermore, the simple power-law parameter tuning is comparable to the complex offline parameter tuning in terms of the solution quality the respective variants produce.

5.2.2 *Statistical Significance.* Table 6 shows a comparison in terms of statistical significance of the best solution found of the fastBRGKA+rev to the existing TSS heuristics as well as our modifications. We notice that our observations from Section 5.2.1 are confirmed. For most of the networks, besides the easy ones, the fastBRKGA+rev is significantly better; in many cases with a very small p-value. This advantage holds also against the fastBRKGA, but it does not exist against the BRKGA+rev. This shows that the reverseMDG heuristic, albeit simple, has a huge impact on the performance of the algorithms.

Table 4 as well as the last column of Table 6 compare the BRKGA variants whose parameters were tuned offline (by López Serrano and Blum [20]) against the respective variants who use the on-the-fly power-law parameter tuning. We see no statistical significance in either case, except for two cases in Table 4, where the fastBRKGA is significantly better than the (tuned) BRKGA. This highlights that an on-the-fly parameter choice following a power-law is comparable to costly offline parameter tuning in this setting.

Table 5: The average run time in seconds of all of the algorithms for which we have run time information. We note that we did not run MMAS and MMAS-Learn (MMAS-L below) ourselves—their respective columns are from Ramírez Sánchez et al. [24, Table 2]. For the BRKGA and the fastBRKGA, the information is from the runs from Table 2. The sizes of the networks are mentioned in Table 2. Bold entries denote the minimum value of a row. We strongly believe that Ramírez Sánchez et al. [24] mixed up the labels of the rows `Dolphins` and `Karate`, which we correct below.

Network	BRKGA	MMAS	MMAS-L	fBRKGA
Dolphins	**< 0.01**	**< 0.01**	**< 0.01**	**< 0.01**
Football	**10.2**	22.0	30.3	21.3
Karate	**< 0.01**	**< 0.01**	**< 0.01**	**< 0.01**
Jazz	12.3	**4.7**	12.9	5.8
CA-AstroPh	**162.2**	173.1	172.6	176.3
CA-GrQc	**51.3**	64.2	72.3	52.1
CA-HepPh	102.90	109.2	110.5	**84.9**
CA-HepTh	88.20	84.6	82.5	**82.3**
CA-CondMat	225.80	**220.4**	222.8	226.2
Email-Enron	328.9	**287.4**	299.6	334.0
ego-facebook	**51.3**	65.7	68.1	54.9
socfb-Brandeis99	**53.9**	68.1	59.3	56.0
socfb-nips-ego	**< 0.01**	**< 0.01**	**< 0.01**	**< 0.01**
socfb-Mich67	**26.1**	68.3	67.5	34.8
soc-gplus	15.30	28.8	**10.2**	44.2
musae_git	**169.4**	202.9	172.7	192.9
loc-gowalla_edges	1 938.5	**728.0**	1 113.5	1 950.5
gemsec_facebook-_artist	457.3	383.6	**330.8**	445.9
deezer_HR	541.6	335.9	**283.9**	535.3
com-dblp	3 059.9	1 749.1	**1 726.5**	3 065.1
Amazon0302	2 527.4	1 768.8	**1 705.4**	2 524.4
Amazon0312	3 820.4	1 747.4	**1 733.1**	3 851.3
Amazon0505	3 993.5	1 746.3	**1 695.6**	3 965.8
Amazon0601	3 862.5	1 726.5	**1 714.0**	3 900.3

5.2.3 *Comparison of the Run Time.* Table 5 shows the average run time in seconds of some of the algorithms we consider. We note that we did not run the MMAS and the MMAS-Learn ourselves but simply report the values from Ramírez Sánchez et al. [24]. Since their setup uses a different machine, the results are not directly comparable. However, since we as well as Ramírez Sánchez et al. [24] ran the BRKGA, we have a common ground for comparison. We find that our run times are close to those reported by Ramírez Sánchez et al. [24]. Hence, we are claim that our results are in general comparable.

We note that the run time of the MMAS and the MMAS-Learn (without the time spent on learning and tuning) is faster than the run time of the fastBRKGA by a factor of around 2. However, the

Table 6: The statistical significance (p-values) of the best solution found of the *fastBRKGA+rev* in comparison to the other shown algorithms. The reported p-values are the result of a Mann–Whitney U test based on the same 10 independent runs per algorithm per network as the ones in Table 2. For MMAS and MMAS-Learn, since we did not run any experiments, we use for each instance the two data points (best and average) from Table 3. Values of statistical significance (that is, with a p-value of at most 0.05) are highlighted in bold. See Section 5.2.2 for more details.

Network	BRKGA	MMAS	MMAS-Learn	fastBRKGA	BRKGA+rev
Dolphins	1.00	1.00	1.00	1.00	1.00
Football	0.63	0.51	0.89	0.96	0.77
Karate	1.00	1.00	1.00	1.00	1.00
Jazz	0.72	0.50	0.91	0.91	0.56
CA-AstroPh	**< 0.001**	**< 0.001**	**0.02**	**0.02**	0.97
CA-GrQc	**< 0.001**	**< 0.001**	0.24	0.33	0.52
CA-HepPh	**< 0.001**	**< 0.001**	0.08	0.08	0.82
CA-HepTh	**< 0.001**	**< 0.001**	**0.02**	**0.02**	1.00
CA-CondMat	**< 0.001**	**< 0.001**	**0.02**	**0.02**	0.26
Email-Enron	**< 0.001**	**< 0.001**	**0.02**	**0.02**	0.12
ego-facebook	**0.02**	**0.001**	**0.02**	**0.02**	0.11
socfb-Brandeis99	0.09	0.37	**0.02**	**0.02**	0.38
socfb-nips-ego	1.00	1.00	1.00	1.00	1.00
socfb-Mich67	0.48	0.52	**0.02**	**0.02**	1.00
soc-gplus	**< 0.001**	**< 0.001**	0.16	0.16	0.37
musae_git	0.48	0.45	**0.02**	**0.02**	0.97
loc-gowalla_edges	**< 0.001**	**< 0.001**	**0.02**	**0.02**	0.62
gemsec_facebook_artist	**0.01**	**0.02**	**0.02**	**0.02**	1.00
deezer_HR	**< 0.001**	**< 0.001**	**0.02**	**0.02**	0.19
com-dblp	**< 0.001**	**< 0.001**	**0.03**	**0.03**	0.16
Amazon0302	**< 0.001**	**< 0.001**	**0.02**	**0.02**	0.60
Amazon0312	**< 0.001**	**< 0.001**	**0.02**	**0.02**	0.33
Amazon0505	**< 0.001**	**< 0.001**	**0.02**	**0.02**	0.73
Amazon0601	**< 0.001**	**< 0.001**	**0.02**	**0.02**	0.62

fastBRKGA does not require any preparations upfront, and its run time is the entire time spent on this algorithms. The MMAS variants were subject to some offline parameter tuning. Thus, the factor of 2 lost in the run time comparison is a fair price to pay.

6 CONCLUSION

We proposed two ways of modifying the BRKGA [20], a state-of-the-art heuristic solver for the target set selection problem (TSS). Our first modification aims to choose the parameter values of the BRKGA during the run instead of tuning them beforehand expensively offline. We choose the value of each parameter with respect to a power-law distribution anew in each iteration. The resulting *fastBRKGA* algorithm yields solutions that are comparable to the highly tuned BRKGA. This shows that the expensive offline tuning used previously can be well replaced by our cheap and easy on-the-fly parameter choice. We believe that such a parameter choice is also a good strategy for other problems than TSS.

Our second modification is adding a simple greedy heuristic specific to the TSS, namely removing vertices from the target set when this does not destroy the target set property. When combined with our power-law parameter choice above, the resulting algorithm significantly outperforms all state-of-the-art algorithms on almost all non-easy instances.

Overall, our results show that the methods applied in the current state of the art, namely, costly parameter tuning and complex computations such as Q-learning combined with a graph convolutional network, are currently not necessary to obtain the best heuristics for TSS problems. Optimizing good solutions greedily and choosing parameter values on the fly are already good by themselves and, in combination, significantly better than the state of the art. We invite other researchers to give both of our approaches a try when they perform their next experiments. Especially, the power-law parameter choice is an easy modification that is applicable to a plethora of randomized search heuristics.

Acknowledgments

This research benefited from the support of the FMJH Program Gaspard Monge for optimization and operations research and their interactions with data science.

REFERENCES

[1] 2024. PACE—Parameterized Algorithms and Computational Experiments. https://pacechallenge.org/. Accessed: 2024-01-29.

[2] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. 2022. Fast mutation in crossover-based algorithms. *Algorithmica* 84 (2022), 1724–1761. https://doi.org/10.1007/s00453-022-00957-5

[3] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. 2024. Lazy parameter tuning and control: choosing all parameters randomly from a power-law distribution. *Algorithmica* 86 (2024), 442–484. https://doi.org/10.1007/S00453-023-01098-Z

[4] Denis Antipov and Benjamin Doerr. 2020. Runtime analysis of a heavy-tailed $(1 + (\lambda, \lambda))$ genetic algorithm on jump functions. In *Parallel Problem Solving From Nature, PPSN 2020, Part II*. Springer, 545–559. https://doi.org/10.1007/978-3-030-58115-2_38

[5] Ning Chen. 2009. On the approximability of influence in social networks. *SIAM Journal on Discrete Mathematics* 23 (2009), 1400–1415. https://doi.org/10.1137/08073617X

[6] Gennaro Cordasco, Luisa Gargano, and Adele A. Rescigno. 2016. On finding small sets that influence large networks. *Social Network Analysis and Mining* 6 (2016), 1–20. https://doi.org/10.1007/s13278-016-0408-z

[7] Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. 2021. Fast immune system-inspired hypermutation operators for combinatorial optimization. *IEEE Transactions on Evolutionary Computation* 25 (2021), 956–970. https://doi.org/10.1109/TEVC.2021.3068574

[8] Duc-Cuong Dang, Anton V. Eremeev, Per Kristian Lehre, and Xiaoyu Qin. 2022. Fast non-elitist evolutionary algorithms with power-law ranking selection. In *Genetic and Evolutionary Computation Conference, GECCO 2022*. ACM, 1372–1380. https://doi.org/10.1145/3512290.3528873

[9] Benjamin Doerr, Yassine Ghannane, and Marouane Ibn Brahim. 2024. Runtime analysis for permutation-based evolutionary algorithms. *Algorithmica* 86 (2024), 90–129. https://doi.org/10.1007/S00453-023-01146-8

[10] Benjamin Doerr, Martin S. Krejca, and Nguyen Vu. 2024. Code and data repository of this paper. https://github.com/nnguyenu/BRKGATSS.

[11] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. 2017. Fast genetic algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2017*. ACM, 777–784. https://doi.org/10.1145/3205455.3205563

[12] Benjamin Doerr and Zhongdi Qu. 2023. A first runtime analysis of the NSGA-II on a multimodal problem. *IEEE Transactions on Evolutionary Computation* 27 (2023), 1288–1297. https://doi.org/10.1109/TEVC.2023.3250552

[13] Benjamin Doerr and Amirhossein Rajabi. 2023. Stagnation detection meets fast mutation. *Theoretical Computer Science* 946 (2023), 113670. https://doi.org/10.1016/j.tcs.2022.12.020

[14] Benjamin Doerr and Weijie Zheng. 2021. Theoretical analyses of multi-objective evolutionary algorithms on multi-modal objectives. In *Conference on Artificial Intelligence, AAAI 2021*. AAAI Press, 12293–12301. https://doi.org/10.1609/AAAI.V35I14.17459

[15] Tobias Friedrich, Andreas Göbel, Francesco Quinzan, and Markus Wagner. 2018. Heavy-tailed mutation operators in single-objective combinatorial optimization. In *Parallel Problem Solving from Nature, PPSN 2018, Part I*. Springer, 134–145. https://doi.org/10.1007/978-3-319-99253-2_11

[16] Tobias Friedrich, Francesco Quinzan, and Markus Wagner. 2018. Escaping large deceptive basins of attraction with heavy-tailed mutation operators. In *Genetic and Evolutionary Computation Conference, GECCO 2018*. ACM, 293–300. https://doi.org/10.1145/3205455.3205515

[17] Jacob Goldenberg, Barak Libai, and Eitan Muller. 2001. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters* 12 (2001), 211–23. https://doi.org/10.1023/A:1011122126881

[18] David Kempe, Jon M. Kleinberg, and Éva Tardos. 2015. Maximizing the spread of influence through a social network. *Theory of Computing* 11 (2015), 105–147. https://doi.org/10.4086/TOC.2015.V011A004

[19] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[20] Albert López Serrano and Christian Blum. 2022. A biased random key genetic algorithm applied to target set selection in viral marketing. In *Genetic and Evolutionary Computation Conference, GECCO 2022*. ACM, 241–250. https://doi.org/10.1145/3512290.3528785

[21] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3 (2016), 43–58. https://doi.org/10.1016/j.orp.2016.09.002

[22] Vijay Mahajan, Eitan Muller, and Frank M. Bass. 1990. New product diffusion models in marketing: A review and directions for research. *Journal of Marketing* 54 (1990), 1–26. https://doi.org/10.1177/002224299005400101

[23] Francesco Quinzan, Andreas Göbel, Markus Wagner, and Tobias Friedrich. 2021. Evolutionary algorithms and submodular functions: benefits of heavy-tailed mutations. *Natural Computing* 20 (2021), 561–575. https://doi.org/10.1007/s11047-021-09841-7

[24] Jairo Enrique Ramírez Sánchez, Camilo Chacón Sartori, and Christian Blum. 2023. Q-Learning ant colony optimization supported by deep learning for target set selection. In *Genetic and Evolutionary Computation Conference, GECCO 2023*. ACM, 357–366. https://doi.org/10.1145/3583131.3590396

[25] Mengxi Wu, Chao Qian, and Ke Tang. 2018. Dynamic mutation based Pareto optimization for subset selection. In *Intelligent Computing Methodologies, ICIC 2018, Part III*. Springer, 25–35. https://doi.org/10.1007/978-3-319-95957-3_4