# The Study of the Target Set Selection Problem under Deterministic Linear Threshold Model Using Evolutionary Algorithms

Mikhail Smirnov, Stepan Kochemazov, Alexander Semenov
ITMO University, St Petersburg, Russia
Emails: masmirnov.inf@gmail.com, stepan.kochemazov@itmo.ru, alex.a.semenov@itmo.ru

***Abstract*—In this paper we study the Target Set Selection problem (TSS) – the well-known combinatorial problem associated with collective behaviour in networks. For a given network graph and the information diffusion model that specifies a dynamic process of how vertices activate each other, TSS aims to identify a set of initially active vertices of minimum cardinality such that they manage to activate all the other network vertices. This problem is computationally hard but has numerous applications in practice, thus the development of computational algorithms for TSS is a relevant topic. A surprising fact is that the spectrum of algorithms for TSS presented so far in the literature is quite limited. The main novelty of our study consists in the hybrid approach which combines evolutionary and greedy algorithms for TSS. In more detail, we view TSS as a pseudo-Boolean optimization problem and solve it using simple evolutionary algorithms (in fact several variants of (1+1)-EA). In order to identify a good initial approximate solution we employ a greedy algorithm. In the experiments we apply the developed algorithms to TSS in the context of Deterministic Linear Threshold Model (DLTM) and demonstrate that the presented approach shows good effectiveness when solving TSS on networks with dozens thousands of vertices on personal computer.**

***Keywords—Boolean satisfiability problem; Target Set Selection; Information Spread***

## I. Introduction

Network analysis is one of the most actively developing areas of modern computer science. It is sufficient to say that the references to key papers on random network models e.g. [1], [2] number in tens of thousands. The present paper studies the phenomenon of information spread in networks and the ways how one can solve combinatorial problems associated with this phenomenon. In particular, we consider the Target Set Selection (TSS) problem [3], [4], which is tightly connected to the Influence Maximization Problem (IMP), which in turn was studied in the widely-cited paper [5].

Both TSS and IMP study the spread of activation in networks, where the vertices represent agents conforming to certain behavioral rules. These rules specify the information diffusion process: how active vertices can activate other vertices at the following time moments. The goal of IMP is for a given network and information diffusion model to find a subset of vertices such that if these vertices are active at the initial time moment, the number of vertices at the end of information diffusion process (the so-called *influence spread*) is maximized. Meanwhile, TSS aims to find a subset of vertices of minimum cardinality such that it activates all vertices in a network.

We consider TSS in the context of the so-called Deterministic Linear Threshold Model (DLTM) (see e.g. [5]), with regards to which from the one hand there are known results about its intractability [3], [4], [5], on the other, for relatively small dimensions it can be solved exactly using some combinatorial algorithms (such as the algorithms for solving the Boolean satisfiability problem (SAT) of Integer Linear Programming problem (ILP)).

The main contributions of the paper are as follows. First, we present a natural formulation of TSS as a pseudo-Boolean black-box optimization problem. Second, we combine evolutionary algorithms for pseudo-Boolean optimization with a greedy algorithm from [6]: the latter is used to find a reasonable starting point for the optimization process. Third, in the computational experiments we show the viability of the proposed approach since it makes it possible to solve TSS under DLTM for networks with tens of thousands vertices on a usual PC.

## II. Preliminaries

Hereinafter, we consider the representation of some collective $V = \{v_1, \dots, v_n\}$ via a directed graph $G = (V, A)$ to which we further refer as to *network*. Here $V$ is the set of vertices of $G$ and $A$ is the set of arcs of this graph. For an arbitrary vertex $v \in V$ let us define its *neighborhood* as a set $U_v \subseteq V \setminus \{v\}$ comprised of all vertices $u$ such that $(u, v) \in A$, i.e. the corresponding arc is directed from $u$ to $v$. For simplicity, we assume that $G$ does not contain loops (however, it is possible to transfer the main results of the paper to take loops into account). Let us associate with network $G$ discrete time moments $t \in \{0, 1, 2, \dots\}$. With each vertex $v \in V$ associate the function $a_v(\cdot)$ the value of which at time moment $t + 1$ ($t \geq 0$) is defined as the function of values of functions $a_u$ for vertices $u \in U_v$ computed at time moment $t$:

$$a_v(t+1) = \begin{cases} 1, & \sum_{u \in U_v} a_u(t) \times w_{(u,v)} \geq \theta_v \\ a_v(t), & otherwise \end{cases} \quad (1)$$

In (1) $w_{(u,v)}$ is some weight (which is usually specified by a positive rational number), and $\theta_v$ is a positive rational number called *threshold* of vertex $v$. In addition to that, we assume that at time moment $t = 0$ the values of all functions of the kind (1) are the numbers from the set $\{0, 1\}$ and that the weights $w_{(u,v)}$ and thresholds $\theta_v$ are specified at time moment $t = 0$ and do not change with $t$.

By $\{0, 1\}^n$ denote the set of all possible binary vectors of length $n$. It is clear, that for an arbitrary $t \in \{0, 1, 2, \ldots\}$ the values of functions (1) for all vertices of network $G$ can be specified by a binary vector $\alpha_t \in \{0, 1\}^n$. Let us refer to this vector as to a *state* of a Discrete Dynamic System (DDS) specified by network $G$ (or simply the state of $G$) at time moment $t$. The described rules specify a function of the kind:

$$F_G : \{0, 1\}^n \to \{0, 1\}^n \tag{2}$$

which defines the transitions between the states of $G$. Let us refer to state $\alpha_0$ of $G$ at time moment $t = 0$ as to *initial* state.

It is clear that for any initial state $\alpha_0$ there exists some $t^*$ such that for all $t > t^*$ it holds that $\alpha_t = \alpha_{t^*}$. Any vector $\alpha \in \{0, 1\}^n$ such that $F_G(\alpha) = \alpha$ is called a *fixed point* of $F_G$. For a considered DDS it is easy to see that starting from any initial state $\alpha_0 \in \{0, 1\}^n$ the corresponding system will reach a fixed point in at most $n$ time moments.

If there are specified functions of the kind (1) for network $G$ and in addition all the rules mentioned above are satisfied, then let us say that the considered DDS is specified under *Deterministic Linear Threshold Model* (DLTM).

We will call vertex $v$ active at time moment $t$, if the coordinate of vector $\alpha_t$ which corresponds to vertex $v$ is equal to 1. A *Target Set* (TS) $V'$, $V' \subseteq V$ is a set of vertices which are active in the initial state $\alpha_0$. Let us define $V'$ by means of some vector $\alpha_0$ and compute the values of functions (1) for some number $s$ of time moments, $s \leq n$. For every vertex $V$ which is active at time moment $t$, $t \leq s$ let us say that $v$ was activated by target set $V'$. Note, that if $\alpha_t$ is not a fixed point, then the number of active vertices at time moment $t + 1$ is greater than that at time moment $t$. Therefore, if at some moment $t$ the number of active vertices is deemed insufficiently small, it makes sense to consider the state $\alpha_{t+1}$.

The *Target Set Selection* problem (TSS) is stated as follows: for the predefined number $R$, $1 \leq R \leq n$ find the smallest target set $V'$ (smallest TS) such that it activates (in the sense above) at least $R$ vertices in a network.

As it is noted in [3] (with reference to [5]) in the formulation described above TSS is NP-hard. Also, based on the results of [5], TSS can not be approximated within any constant factor under assumption that $P \neq NP$. Moreover, TSS is $W[P]$-complete [3] and thus it is fixed-parameter intractable if it is parameterized by the size of the minimum target set. However, TSS can be effectively (in polynomial time in the size of the description of $G$) reduced to combinatorial problems such as SAT or ILP using quite natural reductions, presented e.g. in [7], [8], [9].

## III. TSS as a Pseudo-Boolean Optimization Problem

As we noted above, it is possible to solve both IMP and TSS under DLTM exactly, for example using SAT or ILP, as well as approximately (however, without any guarantees on the approximation ratio). In the present paper we practice the second approach. In particular, we reduce TSS to the pseudo-Boolean optimization problem which we then solve using evolutionary algorithms.

A pseudo-Boolean function (see e.g. [10]) is a function of the following kind:

$$\Phi : \{0, 1\}^n \to \mathbf{R}. \tag{3}$$

The function in equation (3) is not required to be defined analytically. Moreover, it is possible that the values of $\Phi$ are produced by some algorithm, the complexity of which is ignored (essentially, it is treated as an oracle). In such case (3) is referred to as a *black-box function*. If (3) is a black-box function, then one can use only metaheuristic algorithms to optimize it [11]. In this situation it is treated as the so-called *fitness function*.

Note, that when solving TSS the exact methods from [10] or [9] are applicable only to networks with relatively small number of vertices (up to several hundred), since the increase of this number leads to the increase of the encoding size and consequently to longer runtimes of the employed combinatorial algorithms. However, for metaheuristic algorithms this fact is inconsequential, since the optimized function can be computed fast.

The search space in the case of TSS is represented by a Boolean hypercube $\{0, 1\}^n$. An arbitrary $\lambda \in \{0, 1\}^n$ specifies a particular target set: the ones in $\lambda$ correspond to vertices in $G$ which are active at time moment $t = 0$. The fitness function for TSS is the function:

$$\Phi : \{0, 1\}^n \to \{1, \ldots, n\}, \tag{4}$$

which is defined via function (2) as follows: for a particular $\lambda \in \{0, 1\}^n$ first construct the target set represented by the vector corresponding to the initial state $\alpha_0 = \lambda$ of network $G$. Next, compute the functions

$$\begin{gathered} F_G^1(\lambda) = F_G(\lambda), \\ F_G^2(\lambda) = F_G\left(F_G^1(\lambda)\right), \\ \ldots, \\ F_G^k(\lambda) = F_G\left(F_G^{k-1}(\lambda)\right), \end{gathered} \tag{5}$$

where $F_G^k(\lambda) = \gamma$ is either a fixed point of function $F_G$ or $wt(\gamma) \geq \varepsilon \cdot |V|$ ($wt(\cdot)$ is the function computing the Hamming weight), and $\varepsilon \in (0, 1)$ specifies the portion of vertices that need to be active in the original TSS formulation. As we already mentioned above, in order to enter a fixed point it is sufficient to make at most $n$ calculations of the value of function $F_G$ for any TS $\lambda$. The value of $\Phi$ is defined as the Hamming weight of vector $\lambda$, specifying TSS, in the case if $wt(\gamma) \geq \varepsilon \cdot |V|$. Otherwise, if $wt(\gamma) < \varepsilon \cdot |V|$ and $\gamma = F_G^k(\lambda)$ is a fixed point of $F_G$ then the value of (4) in $\lambda$ is viewed as undefined, and the fitness function takes the maximal value $n$.

It is possible to use any metaheuristic algorithm to optimize the function (4) defined above. In the next section we will briefly describe the main algorithms that we will apply to TSS in the formulation above.

## IV. Metaheuristic Algorithms Used to Solve TSS

One of the simplest possible metaheuristic algorithms that can be used to minimize (4) is the well-known (1+1)-Evolutionary Algorithm ((1+1)-EA). Apparently, it was first described in [12]. The elementary step in (1+1)-EA is the so-called random (1+1)-mutation: with an arbitrary vector $\lambda \in \{0,1\}^n$ we associate $n$ random independent Bernoulli trials with probability of success (which is referred to as *mutation rate*) $p = 1/n$. If the trial number $i \in \{1, \dots, n\}$ is successful, then the bit number $i$ in vector $\lambda$ is flipped (e.g. changed into the complementary bit). Let $\lambda'$ be the vector resulting from mutation. If $\Phi(\lambda') \leq \Phi(\lambda)$ (in the case of the minimization problem) then move to the point $\lambda'$ and apply the next random mutation to $\lambda'$, otherwise mutate $\lambda$ again (this situation is called *stagnation*). We always store the Best Known Value (BKV) of the considered fitness function.

The (1+1)-EA algorithm despite its simplicity has a lot of nontrivial mathematical properties [13]. In theory, the algorithm is extremely inefficient: in [13] it was shown that the upper bound on its complexity is $n^n$ in the sense of the expected value of the number of random mutations performed to reach a global extremum, and thus in the worst-case scenario the algorithm is even worse than the simple random search. However, in practice (1+1)-EA can show surprisingly good effectiveness. It follows from the fact that on average only a single bit in $\lambda$ is flipped as a result of random mutation: in more detail, the expected value of the number of flipped bits in $\lambda$ is 1, when the mutation rate is $p = 1/n$. This fact means that on average (1+1)-EA acts similar to the Hill Climbing algorithm [14], and thus can take into account the features of the landscape of a considered function (e.g. it can make use of the convex areas).

A number of works proposed different approaches to the modification of (1+1)-EA aimed at reducing its worst-case complexity. One of the best approaches of this kind is the (1+1)-Fast Evolutionary Algorithm with parameter $\beta$ ((1+1)-FEA$_\beta$), described in [15]. Its main idea lies in the use of the variable mutation rate: in particular, before mutating each $\lambda \in \{0,1\}^n$ a random variable $\mu$ is observed, the spectrum of which is $S_\mu = \{1, 2, \dots, n/2\}$, and the distribution (the so-called "power-law distribution $D^\beta_{n/2}$") is defined as follows

$$\Pr\{\mu = k, k \in S_\mu\} = \left(C^\beta_{n/2}\right)^{-1} \cdot \frac{1}{k^\beta} \tag{6}$$

In (6) $\beta$, $\beta > 1$ is the parameter of the algorithm, $C^\beta_{n/2}$ is a normalizing constant: $C^\beta_{n/2} = \sum_{i=1}^{n/2} i^{-\beta}$ (it is necessary for the normalizing probabilities). A single mutation in the context of (1+1)-FEA$_\beta$ looks as follows: first generate a value $\mu \in \{1, \dots, n/2\}$ in accordance with the distribution $D^\beta_{n/2}$, then apply the random mutation with mutation rate $p = \mu/n$ to the current $\lambda \in \{0,1\}^n$.

The upper bound on the complexity of (1+1)-FEA$_\beta$ is $O\left(C^\beta_{n/2} \cdot n^\beta \cdot 2^n\right)$, which asymptotically is significantly better than that for (1+1)-EA. However, the expected value of flipped bits of (1+1)-FEA$_\beta$ is a constant independent of $n$ only when $\beta > 2$. For example in the case if $\beta = 3$ (see [14]) we have that $E[d_H(\lambda, \lambda')] \approx 1.3685\dots$, where $d_H(\lambda, \lambda')$ is the Hamming distance between vectors $\lambda$ and $\lambda'$).

In paper [16] the switching variant of (1+1)-EA, denoted as (1+1)-SEA$_\delta$ (from Switched Evolutionary Algorithm) was proposed. It simplifies the original idea underlying (1+1)-FEA$_\beta$. This algorithm also employs variable mutation rate, but the switching is performed between two alternatives: the rate of $p = 1/n$ is chosen with probability $1 - \delta/n$, and the rate $p = 1/2$ is chosen with probability $\delta/n$, for parameter $\delta \in (0,1)$. It is easy to show (similar to how it was done in [14]) that for any $\delta \in (0,1)$ the upper bound on the complexity of (1+1)-SEA$_\delta$ of the following kind holds: $O(n \cdot 2^n)$. For the expected value of the number of flipped bits in the case of (1+1)-SEA$_\delta$ we have (see [15]): $E[d_H(\lambda, \lambda')] = 1 + \delta\left(\frac{1}{2} - \frac{1}{n}\right)$. Thus, e.g. for $\delta = 0.5$ the following estimation holds: $E[d_h(\lambda, \lambda')] = 1.25 - o(1)$. Thus, at least theoretically, (1+1)-SEA$_\delta$ looks better than (1+1)-FEA$_\beta$.

Apart from the evolutionary metaheuristics described above we applied to solving TSS the greedy algorithm which is a slightly modified variant of the algorithm from [6]. We also used evolutionary algorithms to start from the initial solution found by the greedy algorithm. The essence of the latter is as follows. Consider network $G$ with functions of the kind (1) associated with vertices. At an arbitrary time moment with vertex $v$ associate two values: $I_v$ which is called the *influence* of vertex $v$ and $\tilde{\theta_v}$ called *residual threshold*. The residual threshold is computed as follows:

$$\tilde{\theta_v} = \theta_v - \sum_{u \in U_v} a_u \times w_{(u,v)}$$

It is clear that if at some time moment $\tilde{\theta_v} \leq 0$ (while at the previous moment it took place that $\tilde{\theta_v} > 0$) then the vertex $v$ becomes active at this time moment.

Assume, that DDS $G$ starts from some TS and goes into a fixed point, and at the same time the constraint on the number of active vertices is not satisfied. In this case we need to choose the next vertex to be included into TS. In accordance with the greedy strategy, the vertex with the greatest influence is added. To compute the influence $I_v$ of vertex $v$ it makes sense to take into account both the number of vertices that can be made active by adding $v$ to TS and how adding it to TS reduced the residual thresholds of the other vertices. Thus, as the measure of influence of vertex $v$ we consider the following value:

$$I_v = |A_v| + \sum_{w \in W} \frac{\Delta\theta_w}{\theta_w} \tag{7}$$

The value of (7) is computed as follows. Assume that before adding $v$ to TS the system $G$ is in the fixed point $\lambda$. We add $v$ to TS and recompute the values of functions (1) for all network vertices until we enter the next fixed point $\lambda'$. Denote by $A_v$ the set of vertices that became active and by $W$ the set of vertices the residual thresholds of which changed during the transition from $\lambda$ to $\lambda'$ (the situation $\lambda = \lambda'$ is allowed).

TABLE I
EXPERIMENTAL RESULTS FOR SMALL NETWORKS. EVOLUTIONARY ALGORITHMS WERE ALLOWED TO PERFORM UP TO $10^5$ ITERATIONS. THE RESULTS OF NONDETERMINISTIC ALGORITHMS ALGORITHMS ARE AVERAGED OVER 10 LAUNCHES AND ARE PRESENTED IN CURSIVE.

| | exact-sat (Glucose3) | greedy | (1+1)EA | greedy&(1+1)EA |
|---|---|---|---|---|
| WS(50,8,0.5)$_{ft}$ | 18 (4093 s) | 26 (0.001 s) | *18.3 (9.93 s)* | *18.3 (10.084 s)* |
| BA(50,4)$_{ft}$ | 14 (1575 s) | 17 (0.007 s) | *14.2 (10.111 s)* | *14 (10.167 s)* |
| WS(50,8,0.5)$_{rt}$ | 7 (3594 s) | 11 (0.008 s) | *7.4 (11.066 s)* | *7.3 (11.119 s)* |
| BA(50,4)$_{rt}$ | 6 (1614 s) | 6 (0.002 s) | *6 (10.973 s)* | *6 (11.145 s)* |

By $\Delta\theta_w$ ($\Delta\theta_w \geq 0$) denote the value by which the residual threshold of $w$ is reduced during $\lambda \to \lambda'$.

After checking all inactive vertices in $\lambda$ we add to TS the vertex $v$ with maximum influence $I_v$.

## V. COMPUTATIONAL EXPERIMENTS

In computational experiments we considered both random networks and fragments of real-world networks. The random networks were constructed in accordance with Watts-Strogatz [1] and Barabasi-Albert [2] models. As for the real-world networks: in particular, we used the 'fb' and 'wiki' networks from the SNAP database [1]. The latter represent the fragments of Facebook (ego-Facebook) and Wikipedia (wiki-vote) respectively. In the original formulation some of the considered graphs (e.g. Watts-Strogatz and Barabasi-Albert) are undirected, thus, we replaced each edge by a pair of arcs with opposite directions, connecting the respective vertices. In all experiments the arc weights were generated as natural numbers chosen randomly and uniformly from $[1, 100]$. As for vertices' thresholds, we considered both the variants where the threshold was specified by a fixed number, as well as ones in which thresholds were generated independently and randomly. In the second case the threshold of an arbitrary vertex $v$ was calculated as a natural number $\theta_v = \lceil \varepsilon \times (\sum_{u \in U_v} w_{u,v}) \rceil$ for $\varepsilon$ randomly and uniformly chosen from $[1/2, 1]$. For the fixed thresholds we used the value of $\varepsilon = 3/4$.

All experiments were carried out on a PC with AMD Ryzen 3950x CPU (16 cores) and 64 GB RAM under Ubuntu OS. The main object of our study are the evolutionary and greedy algorithms for solving TSS that were described above.

For random graphs of small size (several dozen vertices) we checked how the solutions found by metaheuristic algorithms are close to the exact ones. In order to obtain the exact solutions of TSS we employed the algorithms for solving the Boolean satisfiability problem (SAT) [17], which are actively applied today to diverse combinatorial problems. To encode to SAT the problems of finding TSS of size $K$ (i.e. containing $K$ vertices) we employed the methods that were described in detail in [7]. In addition to that we used the SAT encoding techniques for pseudo-Boolean constraints of the kind $\sum_{u \in U_v} a_u(t) \times w_{(u,v)} \geq \theta_v$ from the PySAT toolkit [18]. It should be noted that in [7] only the SAT encodings for the cardinality constraints (e.g. if $w_{(u,v)} \in \{0, 1\}$) were used. To find the smallest $K$ we used the dichotomous scheme that employed the Glucose 3 SAT solver.

The results of these experiments are presented in Table I. In particular, in this table we consider 4 networks with 50 vertices each, generated in accordance with Watts-Strogatz (WS) (with parameters $k = 8$, $\beta = 0.5$) and Barabasi-Albert (BA) (with parameter $m = 4$). We considered both the cases with the fixed threshold for all vertices (e.g. WS(50,8,0.5)$_{ft}$) and

[1]http://snap.stanford.edu

TABLE II
EXPERIMENTAL RESULTS FOR LARGE NETWORKS. EVOLUTIONARY ALGORITHMS WERE ALLOWED TO PERFORM UP TO $10^4$ ITERATIONS. THE RESULTS OF NONDETERMINISTIC ALGORITHMS ALGORITHMS ARE AVERAGED OVER 10 LAUNCHES AND ARE PRESENTED IN CURSIVE.

| | greedy | (1+1)EA | greedy&(1+1)EA | greedy&(1+1)FEA $\beta = 2.5$ | greedy&(1+1)FEA $\beta = 3$ | greedy&(1+1)SEA $\delta = 0.5$ | greedy&(1+1)SEA $\delta = 0.25$ |
|---|---|---|---|---|---|---|---|
| fb | 537 (18.2 s) | *968.7 (320.8 s)* | *437.8 (404.8 s)* | *438.3 (380.6 s)* | *437.1 (380.6 s)* | *440.4 (375 s)* | *438.6 (376 s)* |
| wiki | 3089 (38.5 s) | *3716.6 (220.1 s)* | *3004.7 (347.8 s)* | *3001.7 (320.5 s)* | *3002.5 (321.9 s)* | *3004.6 (319.6 s)* | *3005.3 (319.2 s)* |
| WS(10000,20,0.5) | 2278 (117.6 s) | *3909.7 (353.6 s)* | *2005.3 (722.2 s)* | *1991.0 (658.4 s)* | *1993.6 (661.2 s)* | *2000.0 (667.7 s)* | *2006.1 (667.7 s)* |
| BA(10000,20) | 1103 (341.9 s) | *3891.1 (674.4 s)* | *1073.9 (1356.8 s)* | *1073.1 (1275.6 s)* | *1073.8 (1281.5 s)* | *1072.1 (1274.3 s)* | *1072.0 (1284.7 s)* |

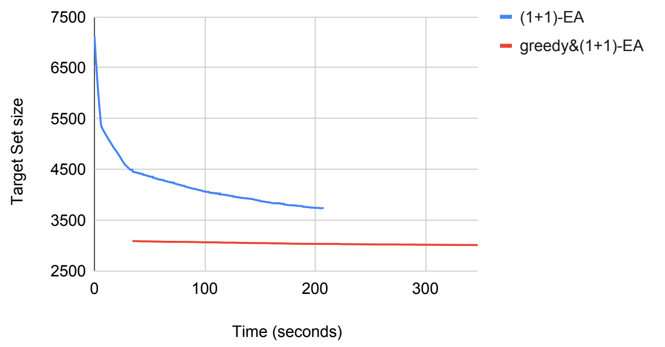

Fig. 1. Convergence plot for a single launch of (1+1)-EA (blue) and of greedy & (1+1)-EA (red) for 10000 iterations on the WIKI problem

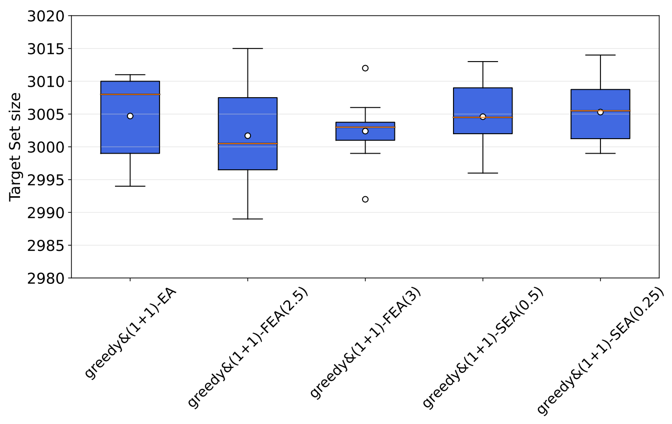

Fig. 2. Box plots of the size of the target set when solving TSS for the WIKI instance using different hybrid algorithms

with thresholds generated randomly according to the scheme outlined above (e.g. BA(50,4)$_{rt}$). The cells of the table present the size of the found TS and the runtime of the employed algorithm. Each evolutionary algorithm in each launch made $10^5$ random mutations. As it was noted above, the SAT solver finds an exact solution, but as it can be seen, even for such small networks, it takes a lot of time. The greedy algorithm works very fast, but the found solutions are not always close to the exact one. As for the (1+1)-Evolutionary Algorithm, it performed $10^5$ random mutations sufficiently faster than the SAT solver and yields the solutions that are close to the exact ones. The best results have been obtained using the mixed strategy, when the greedy algorithm is used to search for an initial solution which is later improved by using (1+1)-EA: in almost all launches this strategy managed to find an exact solution.

In the next series of experiments we considered the networks with several thousand vertices and employed metaheuristic strategies described above, as well as the hybrid strategy combining greedy and evolutionary algorithms, to solve TSS on these networks. Note, that the SAT solvers can not tackle TSS for networks of such dimension.

In Table II we present the benchmarks 'fb' (4039 vertices) and 'wiki' (7115 vertices) mentioned above. We also considered Watts-Strogatz networks (with $k = 20$ and $\beta = 0.5$) and Barabasi-Albert networks (with $m = 20$) with 10 000 vertices. In all cases the thresholds were generated randomly in accordance with the scheme outlined above. Then we applied to them the greedy algorithm as well as several variants of (1+1)-EA, described above. In each launch each evolutionary algorithm was allowed to perform $10^4$ random mutations. What's interesting is that all variants of (1+1)-EA showed more or less the same performance (in the sense of improvement of the found TS). Also note, that the hybrid strategy yielded the best results in the sense of the size of the found TS.

In order to study in detail the behavior of the proposed hybrid algorithm we considered the TSS problem for the WIKI instance in the same configuration as in Table II. We measured the pairs (time, current size of TS) before the first iteration and then after each 100 iterations of the evolutionary algorithm and plotted them at Figure 1. Thus, both red and blue lines have 10000 / 100 = 100 segments. The red plot corresponding to the hybrid greedy&(1+1)-EA algorithm does not start at 0 because the corresponding time is required to construct an initial solution using the greedy algorithm. Also, despite the fact that it seems to not undergo any change, in fact, it steadily decreases from 3089 to 3002.

To analyze the behavior of different variants of hybrid algorithms on average, we performed 20 launches of each algorithm on the same WIKI instance and summed the results of these experiments in the boxplots at Figure 2. Each box represents an interval from 1/4 to 3/4 quartile. The point inside the box is the mean, the line inside corresponds to the median. The whiskers have the length of 1.5 interquartile distance, but since there are only outliers for (1+1)-FEA, in most cases the whiskers just show the interval from the minimum to the maximum. We also performed the Wilcoxon Rank Sum Test between the standard greedy&(1+1)-EA and its four modifications (i.e. 4 tests in total). The hypothesis in each test was that "the target set size in a modified hybrid strategy is smaller than the target set size in greedy&(1+1)-EA" with the significance level $= 0.05$. In all 4 cases the hypothesis was not confirmed.

## VI. Conclusion and Future Works

In this paper we study the well-known Target Set Selection (TSS) problem which is associated with a more general problem of influence maximization [5]. We show that the problem of finding the smallest TSS can be viewed in the context of the general pseudo-Boolean optimization problem. To solve the latter we employ metaheuristic algorithms: greedy algorithm (which is a slightly modified algorithm from [6]) as well as several variants of (1+1)- Evolutionary Algorithm. In the role of benchmarks we use both random graphs (Watts-Strogatz, Barabasi-Albert), and fragments of real-world networks. For such graphs over several thousand vertices (up to 10 000) we demonstrate that the best performance is achieved by the hybrid strategy, in which the greedy algorithm is used to find an initial approximation which is later improved by the evolutionary algorithms. Our plans for the nearest future include the development of parallel variants of evolutionary and genetic algorithms for TSS.

## Acknowledgments

We thank anonymous reviewers for their valuable feedback that made it possible to significantly improve the quality of this paper.

This work was supported by the Ministry of Science and Higher Education of Russian Federation, research project no. 075-03-2020-139/2 (goszadanie no. 2019-1339).

## References

[1] D. J. Watts and S. H. Strogatz, “Collective dynamics of ’small-world’ networks,” *Nature*, vol. 393, pp. 440–442, 1998.

[2] A.-L. Barabási and R. Albert, “Emergence of scaling in random networks,” *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[3] O. Ben-Zwi, D. Hermelin, D. Lokshtanov, and I. Newman, “Treewidth governs the complexity of target set selection,” *Discrete Optimization*, vol. 8, no. 1, pp. 87–96, 2011, parameterized Complexity of Discrete Optimization.

[4] E. Ackerman, O. Ben-Zwi, and G. Wolfovitz, “Combinatorial model and bounds for target set selection,” *Theoretical Computer Science*, vol. 411, no. 44, pp. 4017–4022, 2010.

[5] D. Kempe, J. Kleinberg, and E. Tardos, “Maximizing the spread of influence through a social network,” in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’03, 2003, pp. 137–146.

[6] A. Swaminathan, “An Algorithm for Influence Maximization and Target Set Selection for the Deterministic Linear Threshold Model,” Master’s thesis, Virginia Polytechnic Institute and State University, USA, 2014.

[7] S. Kochemazov and A. Semenov, “Using synchronous boolean networks to model several phenomena of collective behavior,” *PLOS ONE*, vol. 9, no. 12, pp. 1–28, 12 2014.

[8] S. Kochemazov, “Comparative study of combinatorial algorithms for solving the influence maximization problem in networks under a deterministic linear threshold model,” *Procedia Comput. Sci.*, vol. 136, pp. 190 – 199, 2018.

[9] S. Kochemazov and A. Semenov, “Computational study of time constrained influence maximization problem under deterministic linear threshold model for networks with nonuniform thresholds,” in *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2019, pp. 1121–1125.

[10] E. Boros and P. L. Hammer, “Pseudo-boolean optimization,” *Discrete Applied Mathematics*, vol. 123, no. 1, pp. 155 – 225, 2002.

[11] S. Luke, *Essentials of Metaheuristics*, 2nd ed. Lulu, 2013.

[12] H. Mühlenbein, “How genetic algorithms really work: Mutation and hillclimbing,” in *PPSN*, 1992, pp. 15–26.

[13] S. Droste, T. Jansen, and I. Wegener, “On the analysis of the (1+1) evolutionary algorithm,” *Theor. Comput. Sci.*, vol. 276, no. 1–2, p. 51–81, 2002.

[14] S. Russell and P. Norvig, *Artificial Intelligence – A Modern Approach*, 3rd ed. Pearson Education, 2010.

[15] B. Doerr, H. P. Le, R. Makhmara, and T. D. Nguyen, “Fast genetic algorithms,” in *GECCO*, 2017, p. 777–784.

[16] A. Semenov, I. Otpuschennikov, and K. Antonov, “On some variants of the merging variables based (1+1)-evolutionary algorithm with application to maxsat problem,” in *Mathematical Optimization Theory and Operations Research*, ser. LNCS, vol. 12755, 2021, pp. 111–124.

[17] A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds., *Handbook of Satisfiability - Second Edition*, ser. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, vol. 336. [Online]. Available: https://doi.org/10.3233/FAIA336

[18] A. Ignatiev, A. Morgado, and J. Marques-Silva, “PySAT: A Python toolkit for prototyping with SAT oracles,” in *SAT*, 2018, pp. 428–437. [Online]. Available: https://doi.org/10.1007/978-3-319-94144-8_26