Target Set Selection with Constant-Bounded Thresholds Faster Than 2^n

Ivan Bliznets[*] Danil Sagunov[*]

February 18, 2018

Abstract

The Target Set Selection problem formalizes a process of social influence spreading. Given a graph, representing social network, with vertex thresholds, representing numbers of influenced neighbours that vertex needs to become influenced, one is to find a target set, that is, a subset of vertices that is enough to influence pre-specified number of vertices of the network. Influence spreading is defined as an activation process: on the first stage, only the target set is activated, and on each subsequent stage a vertex becomes activated if the number of activated neighbours has reached its threshold. Activated once, a vertex remains activated. Usually it is questioned to find a target set of minimum size that influences the whole social network, so-called minimum perfect target set.

It is known that, even when threshold values of all vertices equals two, the problem is NP-hard, and, from the parameterized point of view, W[P]-hard; it can't be solved in $2^{o(n)}$, unless the Exponential Time Hypothesis fails.

We show that the Target Set Selection problem for an n-vertex graph, where threshold values are bounded by some fixed integer t, can be solved in $\mathcal{O}^*(2^{\lambda_t n})$ time, where $\lambda_t < 1$ depends only on t.

1 Preliminaries

1.1 Problem definition

Consider a graph $G = (V, E)$ and a threshold function $\text{thr} : V \to \mathbb{N} \cup \{0\}$. For a vertex set $X \subseteq V$, we define *activation process yielded by target set X* as a sequence of vertex sets

$$S_0, S_1, S_2, \ldots$$

such that $S_0 = X$ and

[*]Saint Petersburg Academic University

$$S_{i+1} = S_i \cup \{v \in V : |N(v) \cap S_i| \geq \operatorname{thr}(v)\}.$$

If $v \in S_{i+1} \setminus S_i$, we say that v *is activated by* X *in the* $(i+1)^{th}$ *round.* Since all S_i are subsets of V, there is the minimum r such that $S_r = S_{r+1}$. We say that X *activates* S_r and denote it as $\mathcal{S}(X)$. We always omit the graph and the threshold function in this notation since it is always clear from the context.

This leads to a general decision problem.

TARGET SET SELECTION
Input: A graph $G = (V, E)$ with thresholds $\operatorname{thr} : V \to \mathbb{N} \cup \{0\}$, an integers k, l.
Question: Is there a target set of G and thr of size k such that it activates at least l vertices?

If $\mathcal{S}(X) = V$ then we call target set X a *perfect target set.* We call a perfect target set X a *minimum perfect target set* if there is no perfect target set X' such that $|X'| < |X|$. Of course, there can be more than one minimum perfect target set.

Let us give a simple observation that might be useful for better understanding of perfect target set nature.

Observation 1. *Let* $G = (V, E)$ *be a graph and* $\operatorname{thr} : V \to \mathbb{N} \cup \{0\}$ *be a threshold function. If* $X \subseteq V$ *is a perfect target set of* G *and* thr *and* $Y \subseteq V$ *is such that* $\mathcal{S}(Y) \supseteq X$*, then* Y *is a perfect target set of* G *and* thr *too.*

Now we are ready to give a formal definition of the main problem considered in this paper.

MINIMUM PERFECT TARGET SET
Input: A graph $G = (V, E)$ with thresholds $\operatorname{thr} : V \to \mathbb{N} \cup \{0\}$.
Question: Find any minimum perfect target set of G and thr.

Our work focus mainly on instances of MINIMUM PERFECT TARGET SET where threshold function values are bounded by some constant t, i.e. $\operatorname{thr}(V) \subseteq \{0, 1, 2, \dots, t\}$. Even considering $t = 2$ still makes an interesting case. A version of MINIMUM PERFECT TARGET SET, where all threshold values are equal to 2, i.e. $\operatorname{thr} \equiv 2$, was intensively studied and is known under the names of P_3-HULL NUMBER and IRR_2-CONVERSION SET.

Also we always consider only threshold functions that satisfy $\operatorname{thr}(v) \leq \deg(v)$ for any vertex v: otherwise vertex v could be activated no other than being selected in a target set. Thus we can remove vertex v from the graph and reduce the threshold values of all its neighbour vertices by one, what is equivalent to selecting v in a target set. We then solve the problem for the reduced graph and modified threshold function, and simply add v to the resulting perfect target set. Repeating this operation, we can eliminate all the vertices v for which $\operatorname{thr}(v) \leq \deg(v)$ does not hold.

1.2 Algorithm terminology

During its work, our recursive algorithm marks some vertices of the graph as belonging to target set or as not belonging to it. We call such vertices of the graph *fixed* and we call all other vertices (vertices that algorithm has not yet considered belonging to target set or not) *free*.

We use F to denote the set of all fixed vertices and X to denote the set of fixed vertices that were marked as belonging to target set. The set of all free vertices is denoted as $\overline{F} = V \setminus F$. Initially all vertices of the graph are free, therefore $F = X = \varnothing$ initially.

Also we use n for the number of vertices in the graph $|V|$ and k for the number of free vertices $|\overline{F}|$.

2 Algorithm

In this section we provide an algorithm for solving instances of the Minimum Perfect Target Set problem with threshold function values bounded by some fixed integer t. Please consider t as the same constant fixed along the whole section unless stated otherwise.

2.1 Outer part

As many other exponential algorithms, our recursive algorithm is based on the branching technique. Branching here is performed on the set of free vertices, so instance size always equals the number of free vertices. To perform branching, there should be a free vertex v such that v has at least $\mathrm{thr}(v)$ neighbour vertices free. Let us give this branching rule a formal definition.

Lemma 1. *Let $G = (V, E)$ be a graph and $\mathrm{thr} : V \to \mathbb{N} \cup \{0\}$ be a threshold function. Let v be a vertex of G and $T \subseteq N(v)$ be a set of neighbour vertices of v such that $|T| = \mathrm{thr}(v)$. Then there exists a minimum target set X of G and thr such that one of the following conditions stays true:*

- $|X \cap (T \cup \{v\})| < \mathrm{thr}(v)$;
- $X \cap (T \cup \{v\}) = T$.

Proof. Since $|T \cup \{v\}| = \mathrm{thr}(v) + 1$, the only other case is when $|X \cap (T \cup \{v\})| \geq \mathrm{thr}(v)$ and $X \cap (T \cup \{v\}) \neq T$.

Suppose X is such minimum target set and let $Y = (X \setminus (T \cup \{v\})) \cup T$. Note that

$$|Y| = |X| - |X \cap (T \cup \{v\})| + |T| \leq |X| - \mathrm{thr}(v) + \mathrm{thr}(v) = |X|$$

and $X \subseteq Y \cup \{v\}$. Since $|N(v) \cap Y| = |T| = \mathrm{thr}(v)$, $v \in \mathcal{S}(Y)$. The last implies $\mathcal{S}(Y) \supseteq Y \cup \{v\} \supseteq X$. Then by observation 1 Y is a target set of G and thr. Since $|Y| \leq |X|$, Y is a minimum target set, also it satisfies the second condition of the lemma. □

```
Algorithm: tss_outer($G, \mathrm{thr}, F, X$)
Input: Graph $G = (V, E)$, threshold function $\mathrm{thr} : V \to [0, t]$, set of fixed
       vertices $F \subseteq V$, set of fixed targeted vertices $X \subseteq F$.
Output: target set $R$ of $G$ and thr such that $|R \cap F| = X$ and $|R|$ is
        minimum possible; or $V$, if such target set does not exist.
if $\mathcal{S}(X) \subsetneq F$ then
    return tss_outer($G, \mathrm{thr}, F \cup \mathcal{S}(X), X$)
if $\exists v \in \overline{F}$ with $|N(v) \cap \overline{F}| \geq \mathrm{thr}(v)$ then
    $T \longleftarrow$ set of any $\mathrm{thr}(v)$ elements of $N(v) \cap \overline{F}$
    $R \longleftarrow$ tss_outer($G, \mathrm{thr}, F \cup T \cup \{v\}, X \cup T$)
    foreach subset $Y$ of $(T \cup \{v\})$ with $|Y| < |T|$ do
        update $R$ with tss_outer($G, \mathrm{thr}, F \cup T \cup \{v\}, X \cup Y$)
    return $R$
if $k \leq \gamma_t n$ then
    $R \longleftarrow V$
    foreach subset $Y$ of $\overline{F}$ do
        if $X \cup Y$ is a target set then
            update $R$ with $X \cup Y$
    return $R$
else
    return tss_inner($G, \mathrm{thr}, F, X$)
```

Figure 1: Outer part of the algorithm.

The outer part of our algorithm is based on this branching rule and is given in figure 1. It works as follows. Initially it considers all vertices of the graph free, thus none of them were selected to be in target set. It means that the algorithm starts with initial call `tss_outer`$(G, \mathrm{thr}, \varnothing, \varnothing)$. Then, for a set of currently fixed vertices, it ensures that all activated by currently selected target set vertices are fixed. Indeed, according to observation 1 there is no reason to put such vertices in a minimum target set.

Then the algorithm performs the branching itself. To apply the branching rule, algorithm needs a free vertex v with at least $\mathrm{thr}(v)$ neighbours free. If such vertex v is present, it marks v and $\mathrm{thr}(v)$ of its free neighbours as fixed and then it makes recursive calls, each one corresponding to a different selection of these vertices into targeted vertices. These selections are made according to lemma 1, that is, all these vertices will never be selected together as targeted vertices, and, if $\mathrm{thr}(v)$ vertices are targeted, then they always are the neighbours of v. One can find that it makes exactly $2^{\mathrm{thr}(v)+1} - \mathrm{thr}(v) - 1$ recursive calls while fixing $\mathrm{thr}(v) + 1$ vertices: it makes a recursive call for each subset except for the subset of size $\mathrm{thr}(v) + 1$ and $\mathrm{thr}(v)$ subsets of size $\mathrm{thr}(v)$.

The last conditional statement in the outer part stands for the case when no free vertices are activated and the branching rule can not be applied. Please note that the γ_t there stands for some fixed constant, $0 < \gamma_t < 1$, and this constant is explained and specified later.

In this conditional statement, if the set of remaining free vertices is relatively small, algorithm just bruteforces all possible subsets of free vertices to be in the output target set and chooses any of the best of them. Obviously this leads to a correct output. Otherwise, in case free vertices are many, a call to the inner part of the algorithm is performed and it does the remaining work. The main idea is that the inner part is called when there are not many fixed vertices and they satisfy specific conditions, which is further explained in the next section.

2.2 Inner part

2.2.1 Target set compression

Let us summarize the conditions under which the inner part of the algorithm is called:

1. No free vertex is activated by currently selected target set vertices: $\overline{F} \cap \mathcal{S}(X) = \varnothing$.
2. Every free vertex v has less than $\text{thr}(v)$ free neighbour vertices. Since threshold values are bounded by t, it implies that the maximum degree of subgraph induced by set of free vertices is less than t: $\Delta(G[\overline{F}]) < t$.
3. Free vertices make more than γ_t of all vertices: $k > \gamma_t n$, $0 < \gamma_t < 1$.

These conditions follow directly from the corresponding conditional statements of the algorithm outer part.

Our goal is to present a constant $0 < \gamma_t < 1$ and an algorithm such that it works faster than 2^k under conditions above. As we show later, these two are sufficient for the whole algorithm to work faster than 2^n.

The goal of the inner part is to find $X' \subseteq \overline{F}$ so that $\mathcal{S}(X \cup X') = V$. When X' is given, the last equality can be checked trivially. But since k is large, variants of X' are too many. What if we can check $\mathcal{S}(X \cup X') = V$ without actually knowing X' but knowing some specific information about it? We want that information to vary less than X' itself. Indeed, as it shows up, under conditions 1-3, we can specify such information on X'; formally, the following theorem holds.

Theorem 1. *There is a constant $\alpha_t < 1$ and a polynomial-time algorithm $\mathcal{A}$ that, given G, thr, F, X, $\overline{F}$ as usual such that the conditions 1-3 hold, and not given X' explicitly, but by asking questions of the following three types about it:*

Q1. For a free vertex $v \in \overline{F}$, is it true that $v \in X'$?

Q2. For an edge between free vertices $uv \in E(G[\overline{F}])$, is it true that at least one of its endpoints u, v is in X'?

Q3. For a fixed vertex $v \in F$, how many neighbours of v are in X'? If they are more than $\mathrm{thr}(v)$*, answer* $\mathrm{thr}(v)$*.*

$\mathcal{A}$ outputs V if and only if $\mathcal{S}(X \cup X') = V$. Additionally, there are no more than $\mathcal{O}^*(2^{\alpha_t k})$ possible combinations of questions asked by $\mathcal{A}$ and answers to them among all 2^k possible variants of X'. These combinations can be enumerated in $\mathcal{O}^*(2^{\alpha_t k})$ time.

Proof. For simplicity reasons, here we give a proof for the case when $t = 2$. General proof share the same basic idea but requires more details and is given later.

We prove the theorem by giving appropriate algorithm $\mathcal{A}$ and showing that the variants of questions asked by it and answers to them are sufficient.

Of course, questions of the first type are enough for asking to discover X'; though the answers to such questions vary the same as X' itself does. Thus $\mathcal{A}$ should not ask questions of this type much. However, we allow $\mathcal{A}$ to ask questions of the other two types as many as it needs.

In fact, $\mathcal{A}$ simulates the activation process by asking the oracle questions to determine the number of activated neighbours of any vertex with enough precision. As a result, $\mathcal{A}$ finds the activation process yielded by $X \cup X'$, except it misses some vertices of X' possibly. Formally, if $S_0, S_1, \ldots, S_r$ is the activation process yielded by $X \cup X'$, then $\mathcal{A}$ finds a sequence $S'_0, S'_1, \ldots, S'_r$ such that for any $0 \le i \le r$, $S'_i \setminus X' = S_i \setminus X'$.

Since $S_0 = X \cup X'$, S'_0 can be found as $S'_0 = X$. Thus, by induction means, $\mathcal{A}$ has just to find S'_{i+1} when S'_i is found. Precisely, each vertex that is activated in the i^{th} round should be added to S'_{i+1}. It is enough to calculate the number of currently activated neighbours of a vertex v in an assumption that v is not targeted initially, $v \notin X'$. Indeed, if $v \in X'$, any result will suffice, since v can be added to S'_{i+1} or not added to it as well. That calculation is made by asking the oracle questions, the corresponding procedure is presented in figure 2. $\mathcal{A}$ uses this procedure to find S'_{i+1} when S'_i is found and stops when $S'_{i+1} = S'_i$ occurs. As the result, $\mathcal{A}$ returns this S'_i.

Let us prove that $\mathcal{A}$ suffices theorem condition, that is, it finds S'_r such that $S'_r = V$ if and only if $\mathcal{S}(X \cup X') = S_r = V$. $S'_r = V$ implies $S_r = V$ obviously. Suppose then that $S_r = V$, but $S'_r \neq V$, which means that $v \notin S'_r$ for some $v \in X'$. Consider `is_activated` procedure applied to S'_r and v. Since $S_r = V$, $F \subseteq V \setminus X' \subseteq S'_r$, so all fixed neighbours are counted towards activated. In the other hand, all edges with endpoint in v have endpoint in X', so all free neighbours are counted towards activated also. Since we assumed that $\mathrm{thr}(v) \le \deg(v)$, the procedure returns True and then $v \in S'_{r+1}$, which contradicts with $S'_r = S'_{r+1}$.

We are left to analyze the number of questions made by $\mathcal{A}$ to the oracle. Since foreach loop in the `is_activated` procedure considers neighbours of each vertex in the same order, for any fixed vertex v it makes no more than $\mathrm{thr}(v)$ different questions of the first type overall: if $\mathrm{thr}(v)$ questions were made, then v has at least $\mathrm{thr}(v)$ activated neighbours and is activated too without asking

```
Algorithm: is_activated($G$, thr, $F$, $X$, $S'_i$, $v$)
Input: $G$, thr, $F$ and $X$ as usual, $S'_i$ such that $S'_i \setminus X' = S_i \setminus X'$ and a
       vertex $v \notin S'_i$.
Output: True, if $v \notin X'$ and $v \in S_{i+1}$;
        False, if $v \notin X'$ and $v \notin S_{i+1}$;
        any of above, otherwise.
$m \longleftarrow |S'_i \cap N(v) \cap F|$
if $v \in F$ then
    $m \longleftarrow m +$ (Q3) number of neighbours of $v$ in $X'$
    foreach $u \in S'_i \cap N(v) \cap \overline{F}$ in the order they were added to $S'_i$ do
        if $m \geq \mathrm{thr}(v)$ then
            return True
        if (Q1) $u$ is not in $X'$ then
            $m \longleftarrow m + 1$
else
    foreach $u \in N(v) \cap \overline{F}$ do
        if $u \in S'_i$ or (Q2) $uv$ has endpoint in $X'$ then
            $m \longleftarrow m + 1$
return $m \geq \mathrm{thr}(v)$
```

Figure 2: Procedure determining number of activated neighbours of a vertex. (Q1), (Q2), (Q3) precede a question of the corresponding type asked.

any more questions about its neighbours. That gives an upper bound of $t(n-k)$ for the number of questions of the first type asked. This upper bound can be tightened: actually, for any vertex v there is no reason to ask the possible last, $\mathrm{thr}(v)^{th}$ question, since it already means that v has $\mathrm{thr}(v)$ activated neighbours. Such modification gives an upper bound of $(t-1)(n-k)$.

For the number of questions of the other two types asked upper bounds are very simple. Since $\Delta(G[\overline{F}]) < t$, for $t = 2$ the number of edges in $G[\overline{F}]$ is bounded with $1/2k$, therefore the number of different questions of the second type that can be made is bounded with $1/2k$ too. Actually, this upper bound is the only reason we consider the proof for $t = 2$; in case $t > 2$ another upper bound can be proven but it's not such trivial. Still to show the general idea we consider that for an arbitrary t, there is a constant $\omega_t < 1$ such that an $\mathcal{O}^*(2^{\omega_t})$ upper bound is true, so $\omega_2 = 1/2$ but for any $t > 2$ we show that $\omega_t < 1$ exists later.

There are simply $n - k$ possible different questions of the third type. Those require non-binary answer unlike the previous two types though. Since there is no reason to tell the exact count of neighbours if they are already more than t, answers to this questions are non-negative numbers not exceeding t.

Above sums up to

$$2^{(t-1)(n-k)} \cdot 2^{\omega_t k} \cdot (t+1)^{n-k}$$

possible different combinations of questions and answers. Consider

$$(t-1)(n-k) + \omega_t k + \log(t+1)(n-k) \leq \left((t-1+\log(t+1))(\gamma_t^{-1}-1) + \omega_t\right) k$$

and set

$$\alpha_t = (t-1+\log(t+1))(\gamma_t^{-1}-1) + \omega_t$$

and note that we can choose γ_t such that $\alpha_t < 1$. This finishes the proof of the theorem for $t = 2$. $\square$

This theorem provides a possible behaviour for the inner part: iterate over possible combinations of questions and answers; for a fixed combination use $\mathcal{A}$ to check whether it corresponds to X' that $X \cup X'$ is a perfect target set; then find any X' of minimum size corresponding to the fixed combination,. Answer is then chosen as one of the smallest X' found. We now focus on the latter: given a combination as a set of questions and answers, find any appropriate X' of minimum size.

2.2.2 Restoring target sets

Let $F = \{v_1, v_2, \ldots, v_{n-k}\}$ and $\overline{F} = \{u_1, u_2, \ldots, u_k\}$. A combination of questions and answers is fixed, let $d_1, d_2, \ldots, d_{n-k}$ be then the answers to all questions of the Q3-type, i.e. for any i v_i has d_i neighbours in X' if $d_i < \mathrm{thr}(v_i)$ and it has at least $\mathrm{thr}(v_i)$ neighbours in X' if $d_i = \mathrm{thr}(v_i)$.

Once again, for simplicity reasons we first show how to restore a minimum target set satisfying the fixed combination for $t = 2$.

2.2.3 Partial vertex cover families

It appears that, in case $t > 2$, we need a mechanism to both satisfy the theorem 1, specifically to enumerate all possible answers to the Q2-type questions, and provide a possibility to efficiently restore minimum appropriate target set using dynamic programming. The following construction formalizes combinations of answers to the Q2-type questions.

Definition 1. Let $G = (V, E)$ be a graph. We call a subset $S \subseteq V$ of its vertices a *T-partial vertex cover of G* if the edges covered by vertices in S are exactly T:

$$T = \{uv \in E : u \in S \vee v \in S\}.$$

Note that if X' is a T-partial vertex cover of $G[\overline{F}]$, it is enough to know T to answer any Q2-type question about it.

Note that for an arbitrary T-partial vertex cover S, there may be some vertices that can be added to S without changing T. Corresponding definition follows.

Definition 2. Let $G = (V, E)$ be a graph and S be its T-partial vertex cover. Let

$$O_S = \{v \in V \setminus S : N(v) \subseteq S\}$$

and note that for any $O' \subseteq O_S$, $S \cup O'$ is a T-partial vertex cover. We define

$$\mathcal{F}_S = \{S \cup O' \mid O' \subseteq O_S\}$$

and call $\mathcal{F}_S$ a *T-partial vertex cover family of G originated from S.*

Theorem 2. *For any positive integer t, there is a constant $\omega_t < 1$ and an algorithm that, given any n-vertex graph G such that $\Delta(G) < t$, outputs no more than $\mathcal{O}^*(2^{\omega_t n})$ sets $S_1, S_2, \ldots, S_k$, such that the partial vertex cover families originated from it cover all subsets of V, i.e. $\bigcup_{i=1}^{k} \mathcal{F}_{S_i} = \mathscr{P}(V)$, in $\mathcal{O}^*(2^{\omega_t n})$ time.*