Minimum Vertex Augmentation

Jianwen Zhao
Chinese University of Hong Kong
jwzhao@cse.cuhk.edu.hk

Yufei Tao
Chinese University of Hong Kong
taoyf@cse.cuhk.edu.hk

ABSTRACT

This paper introduces a class of graph problems named *minimum vertex augmentation* (MVA). Given an input graph G where each vertex carries a binary color 0 or 1, we want to flip the colors of the fewest 0-vertices such that the subgraph induced by all the (original and new) 1-vertices satisfies a user-defined predicate π. In other words, the goal is to minimally augment the subset of 1-vertices to uphold the property π. Different formulations of π instantiate the framework into concrete problems at the core of numerous applications. We first describe a suite of techniques for solving MVA problems with strong performance guarantees, and then present a generic algorithmic paradigm that a user can instantiate to deal with ad-hoc MVA problems. The effectiveness and efficiency of our solutions are verified with an extensive experimental evaluation.

1 INTRODUCTION

Many graph-based applications endow vertices with binary *states*, such as reliable/unreliable (e.g., sensors), busy/non-busy (delivery depots), customer/non-customer (people), verified/non-verified (intermediate outputs in a workflow), etc. It is often feasible to switch a less desired state to the opposite, e.g., replace a battery's sensor, allocate workers to an understaffed depot, influence a non-customer with advertising, and authenticating an unreliable output. Limited budgets, on the other hand, constrain how many vertices of the bad state can be fixed. Repairing the fewest bad vertices to achieve an optimization objective poses interesting algorithmic challenges.

This paper proposes a *minimum vertex augmentation* (MVA) framework to capture problems of the above style. Consider a directed/undirected graph $G = (V, E)$. Each vertex $v \in V$ has a color, denoted as $color(v)$, which can be 0 or 1. Define V_0 (resp. V_1) as the set of vertices in V having color 0 (resp. 1). We want to find a subset $F \subseteq V_0$ of the minimum $|F|$ such that the subgraph of G induced by $F \cup V_1$ satisfies a user-defined predicate π. This is equivalent to augmenting V_1 by flipping the least 0-vertices to uphold π on the subgraph induced by all the (old and new) 1-vertices. With π formulated differently, the framework defines a class of problems that we call the *MVA class* and denote as $\mathscr{C}_{MVA}$. Some problems in the class extend conventional graph problems in non-trivial manners (as far as algorithm design is concerned), while certain MVA problems do not necessarily have "conventional" counterparts. The next subsection discusses a representative problem selection.

1.1 Representative problems and applications

Problem 1: MVA-shortest-distance. In this problem, G can be directed or undirected. Each edge of G carries a non-negative *weight*. The input also includes a source vertex $s \in V$, a destination vertex $t \in V$, a real value $\tau \geq 0$, and predicate π = "*the subgraph induced by $F \cup V_1$ has a path from s to t whose total length (weight sum of all the edges on the path) is at most τ*".

Consider the graph in Figure 1a, where all edges have weight 1, and the 0- and 1-vertices are shown in white and black, respectively. The shortest path from s to t has length 3. For $\tau = 3$, the optimal solution is to flip (the colors of) a and b. For $\tau = 4$, the optimal solution is to flip only d; note that the *1-path* (i.e., a path consisting of only 1-vertices) s, c, d, e, t is not the shortest path from s to t.

Problem 2: MVA-connectivity. In this problem, G is an undirected graph. The input also includes a set S of *terminal vertices* and predicate π = "*the subgraph induced by $F \cup V_1$ has all the vertices of S in the same connected component*".

Consider the graph G in Figure 1b where $V_1 = \{t_1, t_2, t_3, \mathsf{b}, \mathsf{d}\}$. For $S = \{t_1, t_2, t_3\}$, the optimal solution is to flip $F = \{\mathsf{a}, \mathsf{c}\}$, which connects the terminal vertices with a tree of 6 edges. Note that this is not the smallest tree that can connect t_1, t_2, and t_3; the smallest tree has only 5 edges but requires flipping e, f, and g.

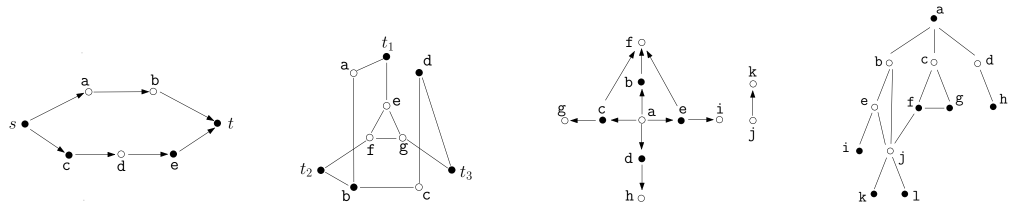
Problem 3: MVA-influence-minseed. Consider G as a social network for viral marketing. We want to promote a product to a *seed set $S \subseteq V$* of people, hoping that they can relay the product information to their acquaintances, who, in turn, may do the same to their acquaintances, and so on. Denote by $spread(S)$ the overall influence of S in this cascading process (the detailed calculation of $spread(S)$ will appear in Section 2). In *MVA-influence-minseed*, the predicate π is "$spread(F \cup V_1) \geq \tau$" for some real value $\tau > 0$.

Figure 1c illustrates the effects of V_1. If V_1 *were* empty, vertex a would make a better seed than j. Given $V_1 = \{\mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}\}$, however, j becomes better. The presence of b, c, d, and e "blocks" a from f, g, h and i such that advertising to a will create no more impact on $\{\mathsf{f}, \mathsf{g}, \mathsf{h}, \mathsf{i}\}$ than what V_1 can *already* achieve. On the other hand, advertising to j may influence one more non-customer, i.e., k.

Problem 4: MVA-reachability. The above problems are not "inherently" MVA because they are meaningful problems even when $V_1 = \emptyset$, i.e., the conventional scenario without the MVA incarnation (see Section 2 for further elaboration). They are better regarded as *MVA extensions* of traditional graph problems. Our next problem, named *MVA-reachability*, is inherently MVA as it makes little sense if $V_1 = \emptyset$. Here, G can be directed or undirected. The input also includes a source vertex s, an integer $\tau > 0$, and the predicate π = "*s can reach at least τ vertices in the subgraph induced by $F \cup V_1$*".

Consider the example in Figure 1d where V_1 contains all the vertices in black. Fix s = a. If $2 \leq \tau \leq 4$, an optimal solution is to flip only vertex c, after which a can reach a, c, f, and g. For $5 \leq \tau \leq 6$, an optimal solution is to flip c and d, allowing a to reach a, c, f, g, d and h. Similarly, an optimal solution for $\tau \in [7, 9]$ is to flip c, d, and j. When $V_1 = \emptyset$, the problem is rather awkward and admits a trivial solution: we can simply return the first τ vertices visited by a DFS or BFS starting from s.

Applications. Unlike the existing problems in the "influence maximization" family (see Section 3), MVA-influence-minseed incorporates the fact that some people will promote a product *voluntarily.*



(a) MVA-shortest-distance (b) MVA-connectivity (c) MVA-influence-minseed (d) MVA-reachability

Figure 1: Illustration of the representative MVA problems (0-vertices in white and 1-vertices in black)

Market mavens, sales-persons, and employees, for example, are naturally motivated to disseminate product information and would form the set V_1. Therefore, resources should focus on people who initially have no incentives to participate in word-of-mouth advertising. The problem's goal is to identify the minimum number of such people to attain a target effect.

MVA-connectivity suggests an alternative marketing strategy also leveraging a social network G. This time, regard V_1 as the set of existing customers. A company can first identify a set S of VIP customers living in the same neighborhood and having similar backgrounds. Based on S, MVA-connectivity outputs a set F of non-customers for promotion. Turning all of F into customers *enforces* a connected community in G (which includes S, F, and some other vertices already in V_1 before). This community has great potentials to attract new customers because the people therein may have common friends who have never patronized the company.

Today, massive sensor networks have become indispensable in smart-city, agriculture, forest protection, etc. Besides the analysis of sensor measurements, a *sensor database* [4, 27, 43] also supports querying sensors' status (e.g., location and battery power). Battery replacement is notoriously tricky. How quickly a battery drains out depends on a sensor's communication volume and its surroundings (in particular, temperature and moisture). By resorting to a sensor database, a modern system monitors every sensor's power and issues replacement jobs as needed to satisfy a performance goal.

MVA-shortest-distance, -connectivity, and -reachability are powerful tools in the above scenarios. The sensors constitute the vertices in a graph G, where an edge exists between two sensors if they are within the communication range; and the edge's weight corresponds to the round-trip time between them. A vertex has color 0 if its sensor's battery has dropped below a certain level, or 1 otherwise. Suppose that vertices s and t represent two critical locations in the sensor network (e.g., two base stations). When they are no longer connected by a robust route (i.e., a fast 1-path), fixing it by replacing the fewest sensors' batteries makes an instance of MVA-shortest-distance. Similarly, in MVA-connectivity, we aim to guarantee robust connectivity among multiple strategic nodes (instead of only 2). Finally, consider s once again as a base station, which ideally should have a 1-path to every sensor. The task of "changing the fewest sensors' batteries for s to maintain 1-paths to at least half of the sensors" makes an instance of MVA-reachability.

The problems defined earlier find further applications in logistics (e.g., "apply the minimum reallocation of the depot workforce to ensure fast package delivery time from location s to location t"), data provenance (e.g., "validate the least intermediate outputs to maximize the number of authenticated facts derived from a data container), urban planning (e.g., "upgrade the fewest road segments to create fast connections among a set of sites"), etc.

1.2 Contributions

This paper studies the MVA class $\mathscr{C}_{MVA}$ at two levels:

- **Micro.** By delving into representative problems, we present generic techniques useful for designing MVA algorithms with attractive performance guarantees.
- **Macro.** By treating $\mathscr{C}_{MVA}$ as a whole, we propose an algorithmic paradigm that can be specialized with user plugins to tackle ad-hoc MVA problems.

Next, we give an overview of our contributions at each level.

Micro: theoretical methods. The selected representative problems adequately illustrate three fundamentally different techniques to approach MVA:

- **Dynamic programming:** The technique allows us to settles MVA-shortest-distance in $\tilde{O}(n + k^*m)$ time[1] where $n = |V|$, $m = |E|$, and k^* is the optimal solution size (same below), i.e., the minimum number of 0-vertices to flip to satisfy π.
- **Half weighting:** MVA-connectivity is NP-hard. We show how to obtain a $(2+|S|/k^*)$-approximate solution in $\tilde{O}(n+m)$ time, after a *half-weighting* transformation that assigns edge weights according to vertex colors.
- **Greedy:** MVA-influence-minseed is NP-hard. We describe a greedy algorithm finding in $\tilde{O}(n + k^*m)$ time a solution F with size $O(1 + k^*)$ and $spread(F \cup V_1) = \Omega(\tau)$.

The above techniques can be combined to approach an MVA problem, which is the case for MVA-reachability. The problem can be shown to be NP-hard. By integrating half weighting and greedy, we obtain an algorithm that runs in $\tilde{O}(|V_0| \cdot (n+m))$ time and finds a solution F larger than the optimal by a factor of $O(k^* \log \tau)$.

Macro: a paradigm. For system engineering, it would be useful to offer a generic paradigm to tackle all MVA problems in a uniform manner. A user can fill in certain details to instantiate our paradigm for different problems. The paradigm returns a high-quality solution fast, continues to search promising areas of the solution space, and produces better answers over time.

2 PROBLEM DEFINITIONS

Consider a directed/undirected graph $G = (V, E)$; define $n = |V|$ and $m = |E|$. We follow the convention that a *directed* edge from vertex u to vertex v is represented as an ordered pair (u, v), while an *undirected* edge between u and v is represented as a set $\{u, v\}$.

[1] The notation $\tilde{O}(.)$ hides a factor polylogarithmic to the problem size.

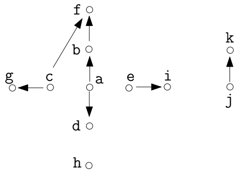


Figure 2: A random graph (MVA-influence-minseed)

Each vertex $v \in V$ has a color in $\{0, 1\}$, denoted as $color(v)$. Define $V_0 = \{v \in V \mid color(v) = 0\}$ and $V_1 = \{v \in V \mid color(v) = 1\}$.

Given a subset U of V, we denote by G_U the subgraph of G induced by U. Let π be a function from $\{G_U \mid U \subseteq V\}$ to $\{\texttt{true}, \texttt{false}\}$. We will be concerned with only monotone π:

Definition 2.1 (Monotonicity). Function π is *monotone* if it has the following property: for any $U_1 \subseteq U_2 \subseteq V$, $\pi(G_{U_1}) = \texttt{true}$ implies $\pi(G_{U_2}) = \texttt{true}$.

Every such π defines an MVA-problem:

Definition 2.2. The *minimum vertex augmentation (MVA) problem* defined by a monotone function π is to find a subset $F \subseteq V_0$ with the smallest $|F|$ such that $\pi(G_{F \cup V_1}) = \texttt{true}$.

We will refer to an F as a *solution* if $\pi(G_{F \cup V_1}) = \texttt{true}$, and an *optimal solution* if $|F|$ is the smallest among all solutions. An MVA-problem is *feasible* if it admits a solution, or *infeasible* otherwise. Note that the problem is infeasible if and only if $\pi(G_V) = \texttt{false}$ due to monotonicity. When $V_1 = \emptyset$ (i.e., all the vertices have color 0), we refer to the degenerated MVA-problem as a *plain problem.*

The MVA-class $\mathscr{C}_{MVA}$ is the set of all MVA-problems defined by monotone functions π.

MVA-shortest-distance. G can be directed or undirected. Each edge $e \in E$ has a *weight* $w(e) \geq 0$. The input specifies a source $s \in V$, a destination $t \in V$, and a value $\tau \geq 0$ such that $\pi(G_U) = \texttt{true}$ if and only if G_U contains a path from s to t whose length is at most τ (the length of a path is the weight sum of its edges).

The problem's plain version finds, among all the s-to-t paths with length at most τ, the one with the fewest vertices. Note that an optimal solution is not necessarily a shortest path from s to t.

MVA-connectivity. G is undirected. The input includes a set S of vertices in V (called the *terminal vertices*) such that $\pi(G_U) = \texttt{true}$ if and only if G_U contains all the terminal vertices in the same connected component.

The plain version returns a tree which (i) is a subgraph of G, (ii) contains all the terminal vertices, and (iii) has the fewest edges among all trees fulfilling the previous two conditions. This is a degenerated version of the *steiner tree problem.*

MVA-influence-minseed. G is directed. The input provides a value $\tau > 0$ such that $\pi(G_U) = \texttt{true}$ if and only if $spread(U) \geq \tau$.

The definition of $spread(U)$ follows the *independent cascade* model [23]. In G, every edge e is given an *influence probability* p_e. Imagine retaining each edge e independently with probability p_e. Denote by E_{rand} the set of edges retained and by $G_{rand} = (V, E_{rand})$ the resulting random graph. Count the number X of vertices v such that at least one vertex in U has a path to v in G_{rand}. The value of $spread(U)$ is the expectation of the random variable X.

<u>Example.</u> Suppose that G is the graph in Figure 1c and that $p_e < 1$ for every $e \in E$ (colors are omitted because they are not needed here). Figure 2 shows one possible G_{rand}. For $U = \{\texttt{a}, \texttt{c}\}$, $X = 6$ because U can reach a, b, c, d, f and g in G_{rand}. □

The plain version of MVA-influence-minseed finds a subset $U \subseteq V$ with the smallest size to satisfy $spread(U) \geq \tau$. This is known as the *minimum target set selection (MINTSS) problem* [14]. It is dual to the *influence maximization problem* [23], where we need to return a set $U \subseteq V$ with the maximum $spread(U)$ subject to $|U| = k$ for some integer $k \geq 1$.

MVA-reachability. G can be directed or undirected. The input gives a source vertex $s \in V$ and an integer $\tau > 0$, such that $\pi(G_U) = \texttt{true}$ if and only if s can reach at least τ vertices in G_U.

The reader can verify that the plain version, although having a sound mathematical definition, is algorithmically trivial and bears little practical significance. MVA-reachability is "inherently MVA", as mentioned in Section 1.1.

3 RELATED WORK

Graph coloring is a general topic in graph theory that studies how to assign colors to vertices/edges to fulfill a designated requirement; see [20, 26] for comprehensive coverage of the existing algorithms. The area's main objective is to understand the minimum number of colors required. In particular, plenty of work has been devoted to the *chromatic number*, the least number of colors necessary to assign different colors to adjacent vertices (see [20] for an excellent survey). On the other hand, the number of colors is fixed to 2 in MVA, whose goal is to understand the minimum number of vertices to *switch* from color 0 to 1 to uphold a property π.

$\mathscr{C}_{MVA}$ should not be confused with *edge switching* problems. In the latter, we have a graph G where each edge is colored either 0 or 1. A *switch operation* selects a vertex u and flips the colors of all the edges incident on u (0 turns into 1 and 1 into 0). The objective is to carry out the operation the least number of times to make a predicate hold (e.g., "the subgraph induced by the 1-edges are connected"). The reader may refer to [16, 17, 19, 31, 35, 45] as entry points into that literature.

We now review results on the plain versions (if they exist) of the MVA problems in Section 2. The input to the *steiner tree problem* includes (i) an undirected graph $G = (V, E)$ where each edge carries a non-negative *weight* and (ii) a set $S \subseteq V$ of *terminal vertices.* Define a *subtree* T of G as a connected acyclic subgraph of G, and the *weight* of T as the sum of the weights of its edges. The problem's goal is to find a subtree of G with the minimum weight that contains all the terminal vertices. The problem is NP-hard [10], with many polynomial-time approximation algorithms known (e.g., [2, 3, 7, 12, 18, 22, 28, 32, 33, 46]). Currently, the best approximation ratio is roughly 1.39, obtained by Byrka et al. [7]. Mehlhorn [28] designed a practical 2-approximate algorithm that runs in $\tilde{O}(n + m)$ time.

The influence maximization problem (see definition in Section 2) was introduced by Kempe et al. [23]. They proved the problem's NP-hardness and gave a $(1 - 1/e - \epsilon)$-approximate algorithm with running time $\Omega(kmn)$. The quest for a faster algorithm has attracted considerable interests [1, 5, 8, 9, 13, 25, 29, 30, 37–39]. A major breakthrough was achieved by Borgs et al [5], whose algorithm runs in time $\tilde{O}(k(m + n))$ and ensures the same approximation ratio as [23]. The closely-related MINTSS problem (Section 2) was studied by Goyal et al. [14]. Based on non-trivial evidence, they conjectured that no polynomial algorithms can find a solution U of size $O(1+k^*)$ where k^* is the size of an optimal solution (we will refer to this as the *MINTSS conjecture*). They also gave an algorithm of $\tilde{O}(k^*(m+n))$ time[2] to find a solution U of size $O(k^*)$ with $spread(U) = \Omega(\tau)$. Our algorithm for MVA-influence-minseed was inspired by the previous work but incorporates new observations necessitated by the presence of V_1. Further details will appear in Section 6.3.

The plain version of MVA-reachability essentially outputs an arbitrary subgraph G' of G such that (i) G' includes the source vertex s, (ii) G' has τ vertices, and (iii) s can reach all the vertices in G'. The problem can be settled in linear time by running any graph traversal algorithm (e.g., DFS and BFS) from s and returning the first τ vertices seen.

$\mathscr{C}_{MVA}$ belongs to the broader class of *combinatorial optimization* where the goal is to find a subset of some universe that minimizes/maximizes a certain objective function. We refer the reader to [24, 34] for a systematic introduction to that grand topic.

4 DYNAMIC PROGRAMMING (MVA-SHORTEST-DISTANCE)

This section will settle MVA-shortest-distance with dynamic programming. It suffices to consider the case where s and t belong to V_1. The assumption does not lose generality because if s (or t) has color 0, it must be added to F (the set of vertices whose colors are flipped) anyway. We will assume that the problem is feasible, as this can be tested in $\tilde{O}(n+m)$ time by computing the shortest path from s to t in the whole G (ignoring colors). We will regard G as a directed graph; if G is undirected, simply replace each edge $\{u, v\}$ with two directed edges (u, v) and (v, u). For an edge $e = (u, v)$, $w(u, v)$ will serve as another representation for the weight $w(e)$.

We use the term *k-zero path* to refer to a path with at most k 0-vertices. For each vertex $v \in V$ and $k \geq 0$, define $spdist(v|k)$ as the minimum length of all k-zero paths P from s to v. We prove in Appendix A:

Lemma 4.1. *$spdist(v|k)$ equals*

$$\begin{cases} \min_{u \in \mathrm{IN}(v)} spdist(u|k) + w(u,v) & \text{if } color(v) = 1 \\ \min_{u \in \mathrm{IN}(v)} spdist(u|k-1) + w(u,v) & \text{otherwise} \end{cases}$$

where $\mathrm{IN}(v)$ *is the set of in-neighbors of* v.

The lemma motivates integrating Dijkstra's algorithm with dynamic programming. At a high level, we maintain a set Q of $dist(v|k)$ values for different $v \in V$ and $k \geq 0$. The value of $dist(v|k)$ is at least $spdist(v|k)$ and decreases monotonically as the algorithm runs. A $dist(v|k)$ is *active* if it is at most τ, or *inactive* otherwise. We impose a total order $\trianglelefteq$ on Q: $dist(v|k) \trianglelefteq dist(v'|k')$ if

- $dist(v|k)$ is active but $dist(v'|k')$ is not;
- (both active) $k < k'$;
- (both active and $k = k'$) $dist(v|k) < dist(v'|k')$.
- (both active, $k = k'$, and $dist(v|k) = dist(v'|k')$) v has a smaller id than v'.[3]

The algorithm essentially inspects values according to $\trianglelefteq$, until encountering the first active $dist(t|k)$ for an arbitrary k:

```
algorithm MVA-shortest-distance
1. insert triplet $(dist(s|0), s, 0)$ into $Q$ where $dist(s|0) = 0$ is the key
2. repeat
3.     remove from $Q$ the triplet $(dist(u|k), u, k)$ with the smallest
       $dist(u|k)$ under $\trianglelefteq$
4.     if $u = t$ then return $k$
5.     for every out-neighbor $v$ of $u$ do relax-edge$(u, v, k)$
subroutine relax-edge$(u, v, k)$
1. if $color(v) = 1$ and $dist(u|k) + w(u, v) < dist(v|k)$ then
   /* convention: $dist(v|k) = \infty$ if $(dist(v|k), v, k)$ is not in $Q$ */
2.     update/insert the triplet $(dist(v|k), v, k)$ in $Q$ with
       $dist(v|k) \leftarrow dist(u|k) + w(u, v)$
3. if $color(v) = 0$ and $dist(u|k) + w(u, v) < dist(v|k+1)$ then
4.     update/insert the triplet $(dist(v|k+1), v, k+1)$ in $Q$ with
       $dist(v|k+1) \leftarrow dist(u|k) + w(u, v)$
```

Example. We illustrate the algorithm on the input graph in Figure 1a, setting $\tau = 4$ (recall that all edges have weight 1). Initially, $dist(s|0) = 0$, whereas $dist(u|k) = \infty$ for every vertex $u \neq s$ and integer $k \geq 1$. After removing $dist(s|0)$ from Q, we perform **relax-edge** on $(s, \mathsf{a}, 0)$ and $(s, \mathsf{c}, 0)$, after which $dist(\mathsf{a}|1) = 1$ and $dist(\mathsf{c}|0) = 1$ are added to Q. As $dist(\mathsf{c}|0) \trianglelefteq dist(\mathsf{a}|1)$, the next entry removed from Q is $dist(\mathsf{c}|0)$, triggering **relax-edge** on $(\mathsf{c}, \mathsf{d}, 0)$, which adds $dist(\mathsf{d}|1) = 2$ to Q. Similarly, $dist(\mathsf{a}|1)$ is then removed; and **relax-edge**$(\mathsf{a}, \mathsf{b}, 1)$ adds $dist(\mathsf{b}|2) = 2$ to Q. The algorithm then removes $dist(\mathsf{d}|1) = 2$, performs **relax-edge**$(\mathsf{d}, \mathsf{e}, 1)$, adds $dist(\mathsf{e}|1) = 3$, removes $dist(\mathsf{e}|1) = 3$, performs **relax-edge**$(\mathsf{e}, \mathsf{t}, 1)$, adds $dist(\mathsf{t}|1) = 4$, removes $dist(\mathsf{t}|1) = 4$, and finishes with $k = 1$. □

We prove the lemma below in Appendix B:

Lemma 4.2. *Algorithm MVA-shortest-distance returns the smallest k^* satisfying $spdist(t|k^*) \leq \tau$. The execution time is in $\tilde{O}(n + k^* m)$.*

Recall that k^* is the size of an optimal solution. We still need to find the path P from s to t corresponding to $spdist(t|k^*)$; namely, P has length $spdist(t|k^*)$ and exactly k^* 0-vertices. After that, the F returned is just the set of 0-vertices in P. To obtain P, it suffices to apply the standard "parent pointer trick" that converts Dijkstra's algorithm from a shortest-*distance* algorithm to a shortest-*path* one. Specifically, we maintain a pointer $parent(v|k)$ for each vertex v and $k \geq 0$. Every time Line 2 (or 4) of **relax-edge** is executed, we set $parent(v|k)$ (or $parent(v|k+1)$, resp.) to u. This introduces only constant-time overhead for Lines 2 and 4 and, hence, does not affect the overall complexity.

5 HALF-WEIGHTING (MVA-CONNECTIVITY)

This section will solve MVA-connectivity with interesting guarantees. We will assume $S \subseteq V_1$ because every 0-vertex in S must be added to F anyway.

5.1 Algorithm

We propose the *half-weighting transformation* that converts the (unweighted) input graph $G = (V, E)$ to a weighted graph G^+ with the same vertex and edge sets. Specifically, to every edge $e = \{u, v\}$, we assign the weight

- 1 if both u and v have color 0,
- 1/2 if only one of u and v has color 0, or

[2] There was no analysis of running time in [14] but one can derive this bound based on the result of [5].

[3] We could do away with this bullet and define $\trianglelefteq$ as a partial order. We feel that it is conceptually cleaner to work with a total order.

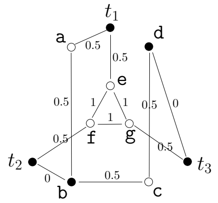


Figure 3: The weighted graph created (MVA-connectivity)

- 0 if neither does.

As another way to understand the above, consider processing each 0-vertex in turn by injecting a weight of $1/2$ to every edge incident on that vertex. This way, an edge incident to two 0-vertices gets injected a total weight of 1.

Our MVA-connectivity algorithm has two steps:

(1) find a minimum steiner tree T^* on G^+ with respect to the terminal vertices in S;
(2) set F to the set of 0-vertices in T^*.

Example. Figure 3 shows the G^+ obtained from the input graph G in Figure 1b. Assuming $S = \{t_1, t_2, t_3\}$, the optimal steiner tree T^* contains vertices t_1, t_2, t_3, a, b, c, and d. □

We cannot execute Step (1) efficiently because the steiner tree problem is NP-hard. Assuming the availability of an α-approximate steiner tree algorithm, we replace T^* in Steps (1) and (2) with an α-approximate steiner tree $\tilde{T}$ of G^+. The modified algorithm ensures:

Lemma 5.1. *We return an F of size at most $k^* \cdot \alpha(1 + \frac{|S|}{2k^*})$, where k^* is the size of an optimal solution to MVA-connectivity.*

The above lemma, which we will prove in Section 5.2, indicates an approximation ratio of $\alpha(1 + |S|/(2k^*))$. By adopting the 2-approximate steiner tree algorithm of [28] (Section 3), we achieve approximation ratio $2 + |S|/k^*$ and running time $\tilde{O}(n + m)$.

It is worth pointing out that MVA-connectivity is NP-hard because the steiner tree problem is NP-hard even when all the edges have unit weight [10] (as mentioned in Section 2, the unit-weight steiner tree problem is the plain version of MVA-connectivity).

5.2 Proof of Lemma 5.1

We start by proving a property of G^+:

Lemma 5.2. *Let T be any tree embedded in G^+ (i.e., each edge of T is in G^+) such that all the leaf nodes of T have color 1. Then, the weight of T is $\frac{1}{2}\sum_{u \in Z} d_T(u)$, where Z is the set of 0-nodes in T, and $d_T(u)$ is the degree of a node u in T.*

Proof. For each node u in T, define $w(u) = 1/2$ if $color(u) = 0$, or 0 otherwise. Given an edge $\{u, v\}$ in T, its weight is precisely $w(u) + w(v)$ by how G^+ is created. Hence, the weight of T equals

$$\sum_{\text{edge } \{u,v\} \text{ in } T} (w(u) + w(v)) \quad = \quad \sum_{\text{node } u \text{ in } T} w(u) \cdot d_T(u)$$

which is $\frac{1}{2}\sum_{u \in Z} d_T(u)$. □

We now establish a connection from minimum steiner trees:

Lemma 5.3. *Let C^* be the weight of an (exact) minimum steiner tree of G^+. It must hold that $C^* < |S|/2 + k^*$.*

Proof. Consider an optimal solution F^* (that is, k^* is the number of nodes in F^*) to the MVA-connectivity problem. $G_{V_1 \cup F^*}$ must contain a stenier tree T with respect to the terminal vertices in S. Let X (or Y) be the set of non-leaf (or leaf, resp.) nodes of T. For sure, $F^* \subseteq X$ and $Y \subseteq S$.

Denote by C the weight of T. Lemma 5.2 tells us:

$$\begin{aligned} C &= \sum_{u \in F^*} \frac{d_T(u)}{2} = \sum_{u \in F^*} \left(\frac{d_T(u)}{2} - 1 + 1 \right) \\ &= |F^*| + \sum_{u \in F^*} (d_T(u)/2 - 1) \le |F^*| + \sum_{u \in X} (d_T(u)/2 - 1) \\ &= |F^*| - |X| + \frac{1}{2} \sum_{u \in X} d_T(u). \end{aligned} \tag{1}$$

The inequality in the above derivation used the fact $F^* \subseteq X$.

T has $|X| + |Y| - 1 = \frac{1}{2}\sum_{u \in X \cup Y} d_T(u)$ edges. Therefore:

$$|X| + |Y| - 1 = \frac{1}{2}\left(\sum_{u \in X} d_T(u) + \sum_{u \in Y} d_T(u) \right) = \frac{1}{2}\left(\sum_{u \in X} d_T(u) + |Y| \right)$$

leading to $\sum_{u \in X} d_T(u) = 2|X| + |Y| - 2$. Thus:

$$(1) = |F^*| + |Y|/2 - 1 < |F^*| + |S|/2$$

where the inequality used the fact $Y \subseteq S$. The claim then follows from $C^* \le C$. □

We now return to the proof of Lemma 5.1. Recall that Step (1) of our (modified) algorithm returns an α-approximate steiner tree $\tilde{T}$ of G^+. Let $\tilde{C}$ be the weight of $\tilde{T}$; by Lemma 5.3, $\tilde{C} \le \alpha(|S|/2 + k^*)$. Since every 0-vertex u in $\tilde{T}$ must be an internal node (otherwise, simply discard u, which strictly reduces the number of 0-vertices in $\tilde{T}$), u has a degree at least 2. By Lemma 5.2, the set F of 0-vertices in $\tilde{T}$ must satisfy $\tilde{C} = \frac{1}{2}\sum_{u \in Z} d_{\tilde{T}}(u) \ge \frac{1}{2}\sum_{u \in F} d_{\tilde{T}}(u) \ge |F|$ where Z is the set of 0-nodes in $\tilde{T}$. We thus have $|F| \le \alpha(|S|/2 + k^*)$, as claimed in Lemma 5.1.

6 GREEDY (MVA-INFLUENCE-MINSEED)

This section will apply the greedy methodology to tackle MVA-influence-minseed. Define:

- $\text{OPT}_{one} = \max_{v \in V} spread(\{v\})$;
- v_{first} as an arbitrary vertex with $spread(\{v_{first}\}) \ge \frac{1}{2}\text{OPT}_{one}$;
- $\tilde{\text{OPT}}_{one}$ as an arbitrary value in $[\frac{\text{OPT}_{one}}{2}, \text{OPT}_{one}]$.

It is possible to find a pair $(v_{first}, \tilde{\text{OPT}}_{one})$ in $\tilde{O}(n + m)$ time with probability at least $1 - 1/n^3$ [5].

We will deal with the following problem instead:

Influence Maximization with Pre-selection (IMP). Given $v_{first} \in V_1$, we want to find a subset $F \subseteq V_0$ with $|F| = k$ to maximize $spread(F \cup V_1)$, where $k \ge 1$ is an integer. Furthermore, we also want to report the value of $spread(F \cup V_1)$.

Let F^* be an optimal solution to IMP. Given an $F \subseteq V_0$ and a real value X, we will call (F, X) a *0.5-approximate solution* if

- $spread(F \cup V_1) \ge 0.5 \cdot spread(F^* \cup V_1)$ and
- $0.5 \cdot spread(F \cup V_1) \le X \le 1.5 \cdot spread(F \cup V_1)$.

The lemma below, proved in Appendix C, connects IMP to MVA-influence-minseed:

Lemma 6.1. *Suppose that we can find a* 0.5*-approximate solution to IMP in* $cost(n, m, k)$ *expected time with probability at least* $1 - 1/n^3$*. Then, for MVA-influence-minseed, with probability at least* $1 - O(1/n^2)$ *we can find in* $\tilde{O}(cost(n, m, k^*))$ *expected time a set* F *of size* $O(1 + k^*)$ *with* $spread(F \cup V_1) = \Omega(\tau)$*, where* k^* *is the size of an optimal solution to MVA-influence-minseed.*

Next, we will focus on finding a 0.5-approximate solution to IMP.

6.1 The reverse-reachability framework

Obtain a random graph G_{rand} from G in the way explained in Section 2. Take a vertex $v \in V$ uniformly at random and identify the set R of vertices that can reach v in G_{rand}. Call this an *experiment* and R a *reverse reachability (RR) set.* It is known [5] that R has size $\tilde{O}(\frac{m}{n}\mathrm{OPT}_{one})$ in expectation and can be obtained in $\tilde{O}(\frac{m}{n}\mathrm{OPT}_{one})$ expected time after some one-off $\tilde{O}(n+m)$ preprocessing time [5, 39]. Repeat the experiment multiple times to produce a collection $\mathcal{R}$ of RR-sets (one per experiment).

Example. Consider the random graph G_{rand} shown in Figure 2. If v is (randomly) selected to be f, the RR set R consists of $\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{f}\}$; if $v = \mathsf{g}$, $R = \{\mathsf{c}, \mathsf{g}\}$; if $v = \mathsf{a}$, then R contains only a. □

Given any $U \subseteq V$, we define X_U as the number of RR-sets $R \in \mathcal{R}$ such that $R \cap U \neq \emptyset$. Borgs [5] proved a lemma — we name the *spread approximation lemma* — stating that $X_U \cdot n/|\mathcal{R}|$ accurately estimates $spread(U)$, as long as $|\mathcal{R}|$ is sufficiently large. The IMP problem thus boils down to finding a subset set $F \subseteq V_0$ to maximize $X_{F \cup V_1}$, which is an instance of the *hitting set problem.*

6.2 Making it work for IMP

The above discussion points to the greedy algorithm below:

1. Remove from $\mathcal{R}$ all the RR-sets R that have a non-empty intersection with V_1.
2. Repeat the following k times with an initially empty F: add to F the vertex u appearing in the largest number of RR-sets in $\mathcal{R}$, and then remove from $\mathcal{R}$ all the RR-sets containing u.
3. Return $(F, X_{F \cup V_1} \cdot n/|\mathcal{R}|)$.

The algorithm can be implemented in time linear to the total size of the RR-sets in $\mathcal{R}$ [5], which is $\tilde{O}(|\mathcal{R}|\frac{m}{n}\mathrm{OPT}_{one})$ in expectation (as mentioned before, each RR-set has an expected size of $\tilde{O}(\frac{m}{n}\mathrm{OPT}_{one})$).

We need to minimize $|\mathcal{R}|$ to maximize efficiency. The analysis of [5] shows that $|\mathcal{R}| = \tilde{O}((k + |V_1|)n/\mathrm{OPT}_{one})$ is a safe upper bound. Our contribution is to improve this bound:

Lemma 6.2. *By setting* $|\mathcal{R}| = \tilde{O}(kn/\mathrm{OPT}_{one})$*, the above greedy algorithm returns a* 0.5*-approximate solution to IMP with probability at least* $1 - 1/n^3$.

Equipped with the lemma, whose proof will be presented in Section 6.3, we can compute a 0.5-approximate IMP solution in $cost(n, m, k) = \tilde{O}(n + m + |\mathcal{R}|\frac{m}{n}\mathrm{OPT}_{one}) = \tilde{O}(n + km)$ time, where the term $\tilde{O}(n + m)$ is due to the one-off preprocessing in Section 6.1.

MVA-influence-minseed is NP-hard.[4] By combining Lemmas 6.1 and 6.2, we can find in $\tilde{O}(n+k^*m)$ time a solution that approximates both the size and influence up to a constant factor. The necessity of such *bi-criteria approximation* is justified by the MINTSS conjecture [14] mentioned in Section 3

6.3 Proof of Lemma 6.2

Greedy on a submodular function. Let us start by considering a general optimization problem. Denote by $\mathbb{D}$ an arbitrary finite set. There is a monotone submodular function $f : 2^{\mathbb{D}} \to \mathbb{R}_{\geq 0}$ (where $\mathbb{R}_{\geq 0}$ is the set of non-negative real values). Given an integer parameter $k \geq 1$, we want to find a set $F \subseteq \mathbb{D}$ to maximize $f(F)$ subject to $|F| = k$. Consider the following greedy algorithm:

1. Repeat the following k times with an initially empty F: add to F the element $u \in \mathbb{D}$ that maximizes $f(F \cup \{u\}) - f(F)$.
2. Return F.

It is well-known that the algorithm is $(1-1/e)$-approximate, namely, $f(F) \geq (1 - 1/e) \cdot f(F')$ where F' is any size-k subset of $\mathbb{D}$.

An approximation scheme. Consider now a similar but (much) more difficult scenario. Given an arbitrary (not necessarily submodular) function $g : 2^{\mathbb{D}} \to \mathbb{R}_{\geq 0}$ and an integer $k \geq 1$, we want to find an $F \subseteq \mathbb{D}$ to maximize $g(F)$ subject to $|F| = k$. Denote by F^* an optimal solution.

The above problem in general admits no fast algorithms. However, an exception arises when g can be approximated by a monotone submodular function $\hat{g} : 2^{\mathbb{D}} \to \mathbb{R}_{\geq 0}$ such that $|g(F) - \hat{g}(F)| \leq \epsilon$ holds for any $F \subseteq \mathbb{D}$ with $|F| \leq k$, where $\epsilon \geq 0$ is a real value. In this case, we can find a $(1 - 1/e)$-approximate F under $\hat{g}$, namely, $\hat{g}(F) \geq (1 - 1/e) \cdot \hat{g}(F')$ for any $F' \subseteq \mathbb{D}$ with $|F'| \leq k$. This F also serves as a good solution under function g:

Lemma 6.3 ([5]). $g(F) \geq (1 - 1/e) \cdot g(F^*) - 2\epsilon$*.*

Proof of Lemma 6.2. We can connect the context of Lemma 6.2 to the above discussion as follows:

- set $\mathbb{D} = V_0$;
- for any $F \subseteq V_0$, define $g(F) = spread(F \cup V_1)$;
- for any $F \subseteq V_0$, define $\hat{g}(F) = X_{F \cup V_1} \cdot n/|\mathcal{R}|$, where $X_{F \cup V_1}$ is the number of sets $R \in \mathcal{R}$ such that $R \cap (F \cup V_1) \neq \emptyset$;
- for any $F \subseteq V_0$, define $f(F) = \hat{g}(F)$.

Lemma 6.4. *By setting* $|\mathcal{R}| = \tilde{O}(kn/\widetilde{\mathrm{OPT}}_{one})$*, with probability at least* $1 - 1/n^3$*, we guarantee*

$$|g(F) - \hat{g}(F)| \leq 0.05 \cdot g(F). \tag{2}$$

simultaneously for all $F \subseteq V_0$ *satisfying* $|F| \leq k$

Proof. We utilize the following fact proved in [5, 39]:

> **Fact:** For any $U \subseteq V$, when $|\mathcal{R}|$ reaches some threshold which is $\tilde{O}(\frac{\ell n}{spread(U)})$, with probability at least $1 - 1/n^{\ell}$ we have $|spread(U) - (n/|\mathcal{R}|) \cdot X_U| \leq 0.05 \cdot spread(U)$.

The definition of IMP ensures $v_{first} \in V_1$. Hence, for any F in the lemma, $g(F) = spread(F \cup V_1) \geq spread(\{v_{first}\}) = \Omega(\mathrm{OPT}_{one})$. Applying the above fact with $\ell = k + c$ (for some constant c decided later) and $U = F \cup V_1$, we know that $|\mathcal{R}| = \tilde{O}(kn/\widetilde{\mathrm{OPT}}_{one})$ suffices to validate (2) for one arbitrary F with probability at least $1 - 1/n^{k+c}$. The number of such F is $\sum_{i=0}^{k} \binom{n}{i} = O(n^k)$. Therefore, (2) holds for all F simultaneously with probability at least $1 - O(n^k/n^{k+c}) \geq 1 - 1/n^3$, as long as constant c is sufficiently large. □

[4] If it was polynomial-time solvable, by setting $V_1 = \emptyset$ we would settle the influence maximization problem with binary search in polylogarithmic time. The latter problem is known to be NP-hard [23].

Let F^* be an optimal solution to the IMP problem. Since $g(F) \leq g(F^*)$, (2) implies $|g(F) - \hat{g}(F)| \leq 0.05 \cdot g(F^*)$. With our formulation of $\mathbb{D}$ and f, the IMP algorithm explained in Section 6.2 is exactly the greedy algorithm presented at the beginning of Section 6.3. Hence, our IMP algorithm finds a $(1 - 1/e)$-approximate F under $\hat{g}$. From Lemma 6.3 (setting $\epsilon = 0.05 \cdot g(F^*)$) and Lemma 6.4, we can conclude that $g(F) \geq (1 - 1/e) \cdot g(F^*) - 2 \cdot 0.05 \cdot g(F^*) > g(F^*)/2$ holds with probability at least $1 - 1/n^3$.

Discussion. A key idea behind our MVA-influence-minseed algorithm is to *force* v_{first} into the returned subset F. Regarding IMP, the proposed algorithm is built on the RR-set technique of [5] and the approximation scheme manifested by Lemma 6.3. Our chief contribution is to observe that IMP can be transformed into submodular optimization in the manner explained in the four bullets under Lemma 6.3. The transformation bounds the number of possible F's with $O(n^k)$, improving the obvious bound of $O(n^{k+|V_1|})$. This is the key behind reducing $|R|$ from $\tilde{O}((k + |V_1|)n/\text{OPT}_{one})$ to $\tilde{O}(kn/\text{OPT}_{one})$.

7 HALF-WEIGHTING + GREEDY (MVA-REACHABILITY)

This section will attack MVA-reachability, which is NP-hard, as shown in Appendix D. By combining the half-weighting (Section 5) and greedy techniques, we will develop an algorithm to return an approximation solution having non-trivial guarantees.

7.1 Algorithm

Our discussion assumes that the source vertex s has color 1; no generality is lost because otherwise s must be added to F anyway.

Let us start by introducing several concepts. Fix a 0-vertex u. We call a subset $S \subseteq V_0$ a *reachability flipping set* of u if flipping S creates a 1-path from s to u (recall that a *1-path* is a path with no 0-vertices). Define $MRF(u)$ as a reachability flipping set of u with the minimum size; if there are multiple such reachability flipping sets, $MRF(u)$ can be any of them.

Example. Consider the 0-vertex j in Figure 1d where the source vertex is a. {j, e, b}, {j, c}, and {j, b} are all reachability flipping sets of j. MRF(j) can be either {j, c} or {j, b}. □

The next two concepts are *semi-direct 1-reachability* and *direct 1-reachability.* Fix a 0-vertex u and a 1-vertex v. We say that u can *semi-directly 1-reach* v if G contains a path from u to v on which all the vertices have color 1 except u. Fix, instead, a 1-vertex u and a 1-vertex v. We say that u can *directly 1-reach* v if G contains a 1-path from u to v.

Example. In Figure 1d, the 0-vertex j can semi-directly 1-reach {k, l, f, g}, while the 1-vertex f can directly 1-reach {f, g}. □

The last concept we will need is *coverage.* If u is a 0-vertex, we define $coverage(u)$ to be the set of 1-vertices semi-directly 1-reachable from u. Otherwise, $coverage(u)$ is the set of 1-vertices directly 1-reachable from u. Given a subset $S \subseteq V$, we define $coverage(S) = \bigcup_{u \in S} coverage(u)$.

Example. In Figure 1d, $coverage$(j) = {k, l, f, g} and $coverage$(f) = {f, g}. Hence, $coverage$({j, f}) = {k, l, f, g}. □

Let us assume that we have obtained $MRF(u)$ for every $u \in V_0$ and $coverage(u)$ for every $u \in V_0 \cup \{s\}$ (how to achieve this will be discussed later). We now formulate an instance of the *partial set cover* problem. Given any $S \subseteq V_0$, define $cost(S) = \sum_{u \in S} |MRF(u)|$. S is a *solution set* if $|coverage(S \cup \{s\})| \geq \tau$. The partial set cover instance aims to return a solution set S of the minimum $cost(S)$. The problem is NP-hard but the standard greedy algorithm returns a solution set whose cost is larger than the optimum by a factor of $O(\log \tau)$ [42]. The running time is $\tilde{O}(\sum_{u \in V_0 \cup \{s\}} |coverage(u)|)$.

Let S be the output of the greedy algorithm. We return $F = \bigcup_{u \in S} MRF(u)$ for MVA-reachability. Section 7.2 will prove:

Lemma 7.1. *If k^* is the size of an optimal solution to MVA-reachability, the above algorithm finds an F whose size is larger by a factor at most $O(\log \tau) \cdot k^*$.*

Computing *coverage*(u) and *MRF*(u). We can obtain $coverage(u)$ by performing a DFS on the subgraph induced by $V_1 \cup \{u\}$; the running time is $O(n + m)$ (where $n = |V|$ and $m = |E|$). To compute $MRF(u)$, we resort to the half-weighting idea, as explained next.

Construct from G the weighted graph G^+ in the way shown in Section 5.1. Let P be a path from the source vertex s to a 0-vertex u. As a corollary of Lemma 5.2, the length of P in G^+ (i.e., the total weight of the edges in P) plus 1/2 is precisely the number of 0-vertices on P. Thus, if P is a shortest path from s to u on G^+, the set of 0-vertices on P can be taken as an $MRF(u)$. Therefore, we can obtain $MRF(u)$ using Dijkstra's algorithm in $\tilde{O}(n + m)$ time.

Summary. In $\tilde{O}(|V_0|(n+m))$ time, we can acquire $coverage(u)$ and $MRF(u)$ for all $u \in V_0 \cup \{s\}$. The complexity dominates the time of solving the partial set cover instance as $|coverage(u)| \leq n$ for all u.

7.2 Proof of Lemma 7.1

Denote by F^* an optimal solution to MVA-reachability. We know:

Lemma 7.2. *F^* is a solution set to the partial set cover instance.*

Proof. By definition of F^*, after flipping the vertices in F^*, s has 1-paths to at least τ 1-vertices v. In G, every such v must be either directly 1-reachable from s or semi-directly 1-reachable from a 0-vertex in F^*. Hence, $coverage(F^* \cup \{s\}) \geq \tau$. □

Lemma 7.3. *$cost(F^*) \leq |F^*|^2$.*

Proof. It suffices to prove that $|MRF(u)| \leq |F^*|$ for every $u \in F^*$. The claim will then follow from $cost(F^*) = \sum_{u \in F^*} |MRF(u)|$.

Assume, on the contrary, that $|MRF(u)| > |F^*|$. The definition of $MRF(u)$ implies that s cannot reach u with a 1-path even after flipping all the vertices in F^*. Thus, removing u from F^* will not reduce the number of vertices reachable from s in $G_{V_1 \cup F^*}$. This contradicts the optimality of F^* for MVA-reachability. □

Recall that S is the solution set to the partial set cover instance found by our algorithm, and F is the algorithm's output for MVA-reachability. Lemma 7.2 indicates that $cost(S) = O(\log \tau) \cdot cost(F^*)$. Regarding F, we have $|F| = |\bigcup_{u \in S} MRF(u)| \leq \sum_{u \in S} |MRF(u)| = cost(S)$. We thus conclude that $|F| \leq cost(S) = O(\log \tau) \cdot cost(F^*) \leq O(\log \tau) \cdot |F^*|^2$, where the last inequality used Lemma 7.3. This completes the proof of Lemma 7.1.

8 A GENERIC FRAMEWORK

This section will describe an algorithmic paradigm that can be instantiated to approach ad-hoc MVA problems. The paradigm aims to output a good solution F quickly and then look for better solutions by searching a promising portion of the solution space.

8.1 The paradigm

Encoding the solution space. As before, denote by π the monotone condition that defines the underlying MVA problem. Recall that a *solution* refers to a set $F \subseteq V_0$ such that $\pi(G_{F \cup V_1}) = \texttt{true}$.

We use a *solution tree* $\mathscr{T}$ to encode all possible solutions. $\mathscr{T}$ is a perfect binary tree with $2^{|V_0|}$ leaves. Every edge of $\mathscr{T}$ is *labeled* with (i) a vertex in V_0 and (ii) a status: *alive* or *dead*. Two requirements are enforced for each internal node z:

- the left and right edges of z must be labeled with the same vertex, called the *branching vertex* of z;
- the left edge must be alive and the right one must be dead.

For each node z in $\mathscr{T}$, denote by A_z (or D_z) the set of alive (or dead, respectively) vertices on the path from the root of $\mathscr{T}$ to z. We demand that the A_z of all leaf nodes z be distinct; in other words, each A_z corresponds to a unique subset of V_0.

Example. Figure 4 illustrates one possible solution tree on $V_0 = \{\texttt{a}, \texttt{b}, \texttt{c}\}$. Node z_5, for example, has branching vertex c, $A_{z_5} = \{\texttt{b}\}$, and $D_{z_5} = \{\texttt{a}\}$. □

The *next* function. We can characterize $\mathscr{T}$ using a simple function. Specifically, define $next(A, D)$ as a function which takes *disjoint* subsets A, D of V_0 as parameters, and outputs a vertex in $V_0 \setminus (A \cup D)$. To see how the function decides a solution tree $\mathscr{T}$, imagine growing $\mathscr{T}$ in a top-down manner: after creating an internal node z, we set $v = next(A_z, D_z)$ as the branching vertex of z, which uniquely generates the left and right children of z. The procedure below formalizes this process:

```
algorithm build-solution-tree $(A, D)$
    if $A \cup D = V_0$ then return NULL
    $v = next(A, D)$
    $\mathscr{T}_1 \leftarrow$ build-solution-tree $(A \cup \{v\}, D)$
    $\mathscr{T}_2 \leftarrow$ build-solution-tree $(A, D \cup \{v\})$
    create a node $z$ with $\mathscr{T}_1$ (or $\mathscr{T}_2$) as the left (or right) subtree
        and $v$ as the branching vertex
    return the tree rooted at $z$
```

To construct $\mathscr{T}$, simply call **build-solution-tree**$(\emptyset, \emptyset)$.

Example. Consider again the solution tree in Figure 4. Judging from the fact that z_0 has branching vertex b, we know $next(\emptyset, \emptyset) = \texttt{b}$. Likewise, it can be inferred from node z_1 that $next(\{\texttt{b}\}, \emptyset) = \texttt{a}$, and from z_8 that $next(\emptyset, \{\texttt{b}\}) = \texttt{c}$. □

The $next(A, D)$ function lends itself nicely to the greedy methodology. Imagine building a solution incrementally by adding one vertex at a time. Treating (i) A as the set of 0-vertices whose colors will be flipped, and (ii) conversely D as the set of vertices whose colors will not be flipped, we ask the question: *under such circumstances, which should be the next vertex to flip?* The answer is precisely the output of $next(A, D)$.

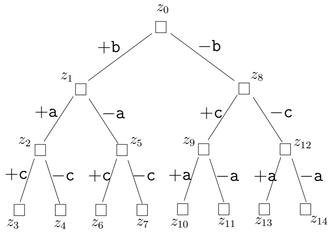


Figure 4: A possible solution tree on $V_0 = \{\texttt{a}, \texttt{b}, \texttt{c}\}$

Traversing the solution tree. Fix a function $next(A, D)$ (i.e., a greedy strategy) and thus also a solution tree $\mathscr{T}$. We perform a *pre-order* traversal of $\mathscr{T}$. For each node z encountered, check whether A_z is a solution; if a better solution is found, retain it. A user can terminate the traversal when s/he is happy with the current solution. The process can be accelerated by leveraging:

- **(Fact 1)** If $\pi(G_{A_z \cup V_1}) = \texttt{true}$, the subtree of z can be pruned, because $A_{z'}$ cannot have fewer 0-vertices than A_z for any descendant z' of z.
- **(Fact 2)** Let F_{now} be our best solution so far. If A_z contains at least $|F_{now}|$ 0-vertices, the subtree of z can be pruned.
- **(Fact 3)** Consider z as the right child of its parent. If $\pi(G_{(V_0 \setminus D_z) \cup V_1})$ $= \texttt{false}$, the subtree of z can be pruned. No descendant z' of z can yield a solution $A_{z'}$ due to monotonicity.

Linear space implementation. Our paradigm requires only $O(n)$ space to implement (materialization of the whole $\mathscr{T}$ is unnecessary). The main observation is that a node z in $\mathscr{T}$ can be *uniquely identified* using the labels on the root-to-z path, which can be stored with $O(n)$ space. The path contains all the information we need to carry out the pre-order traversal. Specifically, to descend into the left child of z, we obtain the branching vertex v of z by calling $next(A_z, D_z)$. The vertex v is kept at z. This occupies extra space, but when there is a need to descend into the right child of z, we can do so directly without another call to $next(A_z, D_z)$.

Searching a polynomial core. The solution space can remain large even with the pruning by Facts 1-3. To limit the search to a good region in the solution space, we propose:

(Bounded-death heuristic) Prune the subtree of a node z in $\mathscr{T}$ if $|D_z| > \lambda$, where $\lambda \geq 0$ is a constant.

Our algorithm, in general, first descends the leftmost path of $\mathscr{T}$ until finding the first node z such that A_z is a solution. Setting $k_{first} = |A_z|$, we prove in Appendix E:

Lemma 8.1. *At most $(k_{first} + 1)^{\lambda+1}$ nodes in $\mathscr{T}$ are visited.*

We call the set of nodes counted in the lemma the *polynomial core* of the *next* function. An effective *next* function should lead us quickly to a first solution. The polynomial core represents a small region "around" that solution, which our algorithm scrutinizes to look for better solutions. It should be noted that the lemma does not imply solving MVA in polynomial time because our processing framework still requires exponential time if an optimal solution must be found. Searching the polynomial core is a heuristic that aims to improve the first solution within a user's time constraint.

As a remark, Facts 1-3 have different importance. Facts 1-2 are compulsory because they are needed in the proof of Lemma 8.1. Fact 3, on the other hand, is optional and helps improve the efficiency by roughly 5% based on our empirical evaluation.

8.2 Paradigm instantiation for Problems 1-4

To solve an MVA-problem, a user needs to implement two *plug-ins*:

- the *next*(A, D) function introduced in Section 8.1 and
- the $\pi(G_{A\cup V_1})$ function, i.e., testing whether A is a solution.

Next, we demonstrate how to do so for Problems 1-4.

MVA-shortest-distance. The function $\pi(G_{A\cup V_1})$ returns `true` if and only if the shortest path from the source s to the destination t has length at most τ in $G_{A\cup V_1}$. Regarding *next*(A, D), our objective is to find a 0-vertex $v \notin (A \cup D)$ to induce a path $s \rightsquigarrow v \rightsquigarrow t$ such that (i) the path has length at most τ, (ii) the intermediate vertices on $s \rightsquigarrow v$ are all in $A \cup V_1$, and (iii) the part $v \rightsquigarrow t$ has few 0-vertices. Towards this purpose, define $sp_{\overline{D}}(v, t)$ as an arbitrary shortest path in G, among all the paths from v to t that do not pass any vertex in D. Let $spdist_{\overline{D}}(v, t)$ be the length of $sp_{\overline{D}}(v, t)$, and $zcnt_{\overline{D}}(v, t)$ the number of 0-vertices on $sp_{\overline{D}}(v, t)$. Call a 0-vertex $v \notin (A \cup D)$ a *candidate* if v has an in-neighbor $u \in (A \cup V_1)$ satisfying $spdist_{A\cup V_1}(s, u) + w(u, v) + spdist_{\overline{D}}(v, t) \le \tau$ where $spdist_{A\cup V_1}(s, u)$ is the shortest path distance from s to u in $G_{A\cup V_1}$. Function *next*(A, D) returns the candidate vertex v with the smallest $zcnt_{\overline{D}}(v, t)$.[5]

MVA-connectivity. Let $s_1, s_2, ..., s_{|S|}$ be the terminal vertices in S (arbitrary ordering). Our strategy is to connect them incrementally: first find a path to link up s_1 and s_2, extend the path into a tree which includes s_3, then s_4, and so on, while striving to minimize the number of 0-vertices in the process.

The function $\pi(G_{A\cup V_1})$ returns `true` if and only if $G_{A\cup V_1}$ contains all terminal vertices in one connected component (CC). To explain the *next* function, let $x \ge 1$ be the smallest integer such that $G_{A\cup V_1}$ includes $s_1, ..., s_x$ in the same CC. Call $v \in V_0$ a *candidate* if v (i) does not belong to $A \cup D$, (ii) is outside the CC, and (iii) is adjacent to at least one vertex in the CC. We define *next*(A, D) to be the candidate vertex v with the smallest $zcnt_{\overline{D}}(v, s_{x+1})$, where $zcnt_{\overline{D}}(v, s_{x+1})$ has the same meaning as in MVA-shortest-distance.

MVA-influence-minseed. Our algorithm always adds the vertex v_{first} described in Section 6 to V_1. We implement functions *next* and π by resorting to the spread approximation lemma (Section 6) and RR-sets of different sizes. A small RR-set $\mathcal{R}_{next}$ suffices for *next* because the function is only for greedy selection and does not affect the correctness of solutions. The RR-set $\mathcal{R}_\pi$ for π, however, must have a decent size for the ultimate quality control. Specifically, in *next*(A, D), we first remove all the RR-sets $R \in \mathcal{R}_{next}$ such that $R \cap (A \cup V_1) \ne \emptyset$; let $\mathcal{R}'_{next}$ be the set of remaining RR-sets. The function returns the 0-vertex $v \notin A \cup D$ that appears in the largest number of RR-sets in $\mathcal{R}'_{next}$. Function $\pi(G_{A\cup V_1})$, on the other hand, returns `true` if and only if $X \cdot n/|\mathcal{R}_\pi| \ge \tau$, where X is the number of RR-sets in $\mathcal{R}_\pi$ that intersect with $A \cup V_1$.

While $|\mathcal{R}_{next}|$ is set to a constant (10000 in our experiments), we adopt an incremental approach for the size of $\mathcal{R}_\pi$. To ensure success probability at least $1 - 1/n$, (as a corollary of Lemma 6.4) we only need to set $|\mathcal{R}_\pi| = \tilde{O}(\log n_{call} \cdot n/\tilde{\text{OPT}}_{one})$ where n_{call} is the number of calls to the π function, and $\tilde{\text{OPT}}_{one}$ is obtained together with v_{first} (Section 6). This way, $|\mathcal{R}_\pi|$ only needs to increase slowly with time. We achieve the purpose by appending new RR-sets to $\mathcal{R}_\pi$ as needed during the execution.

MVA-reachability. The function $\pi(G_{A\cup V_1})$ returns `true` if and only if the source vertex s can reach at least τ vertices in $G_{A\cup V_1}$. To implement *next*(A, D), we define a *candidate* as a 0-vertex v satisfying the following conditions. First, $v \notin A \cup D$. Second, v is adjacent in G to at least one vertex that s can reach in $G_{V_1\cup\{A\}}$ with a 1-path. The function *next*(A, D) returns the candidate v with the largest *coverage*$(v) \setminus$ *coverage*$(A \cup \{s\})$ (see Section 7.1 for the definition of coverage).

9 EXPERIMENTS

Data graphs. Our evaluation was based on four graphs[6]: (i) *DBLP* (undirected, $n = 317080$, $m = 1049866$) has an edge between two researchers if they have co-authored a paper; (ii) *Youtube* (undirected, $n = 1134890$, $m = 2987624$) has an edge between two persons if they are friends on Youtube; (iii) *Wiki* (directed, $n = 2394385$, $m = 5021410$) has an edge from a user to another if the former has ever messaged the latter in Wikipedia; and (iv) *USpatent* (directed, $n = 3774768$, $m = 16518947$) has an edge from a patent to another if the former's legal document cites the latter.

Different preprocessing was carried out for each problem:

- The input of MVA-shortest-distance is a weighted graph. We generated edge weights based on random walks. The generation started by calculating the page rank $\rho(v)$ of each vertex v. If G was directed, we set the weight of edge (u, v) to $\rho(u)/d^+(u)$ where $d^+(u)$ is the out-degree of u; otherwise, the weight of edge $\{u, v\}$ was $\rho(u)/d(u) + \rho(v)/d(v)$ where $d(u)$ is the degree of u.
- MVA-connectivity takes an undirected graph as the input. For *Wiki* and *USpatent*, we obtained their undirected versions by ignoring the edge directions.
- MVA-influence-minseed requires a directed graph. We converted *DBLP* and *Youtube* into directed graphs by replacing each edge $\{u, v\}$ with two directed edges (u, v) and (v, u). Following a common approach in the literature of influence maximization [8, 13, 21, 38, 41], we set the influence probability p_e (Section 2) of an edge $e = (u, v)$ to $1/d^-(v)$ where $d^-(v)$ is the in-degree of v.

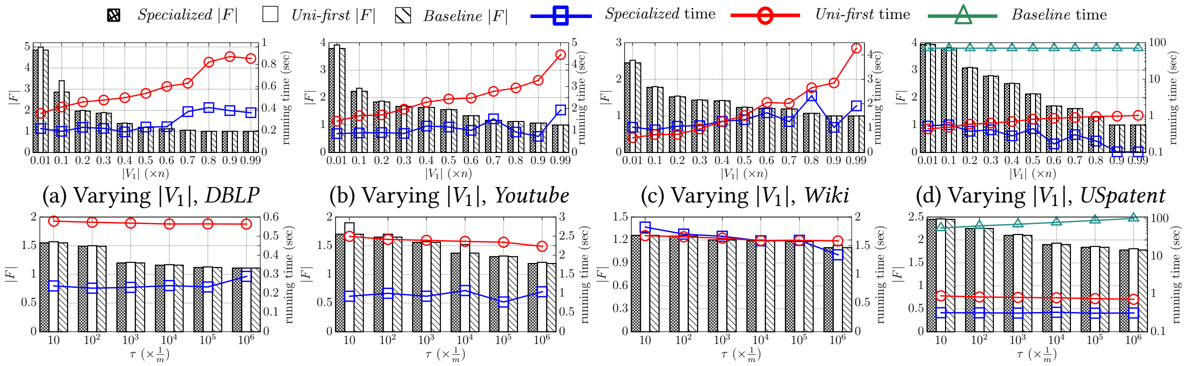
No special preprocessing was necessary for MVA-reachability.

Competing methods. Recall that Sections 4-7 have presented an algorithm for each representative MVA problem. We will call those algorithms collectively as *Specialized* (no ambiguity can arise as long as the problem context is clear). We will use *Universal* to refer to the algorithmic paradigm in Section 8.1, integrated with the plugins described in Section 8.2. Particular attention will be paid to *Universal*'s first solution because its quality and production time indicate the effectiveness and efficiency of the *next* function. We will use the name *Uni-first* for the special version of *Universal* that terminates right after the first solution.

For benchmarking, we also experimented with a *Baseline* method for each problem. For MVA-shortest-distance, *Baseline* is the algorithm of [44] which reports the paths from a source vertex s to a destination vertex t in ascending order of length. The algorithm finishes at discovering a path with length greater than τ, and solves MVA-shortest-distance optimally by returning the path having the fewest 0-vertices. For MVA-connectivity, *Baseline* is an implementation of [3, 12] which solves the steiner-tree problem with

[5] If more than one candidate fulfills the condition, *next*(A, D) returns an arbitrary one. The same policy applies in the *next* functions of Problems 2-4.

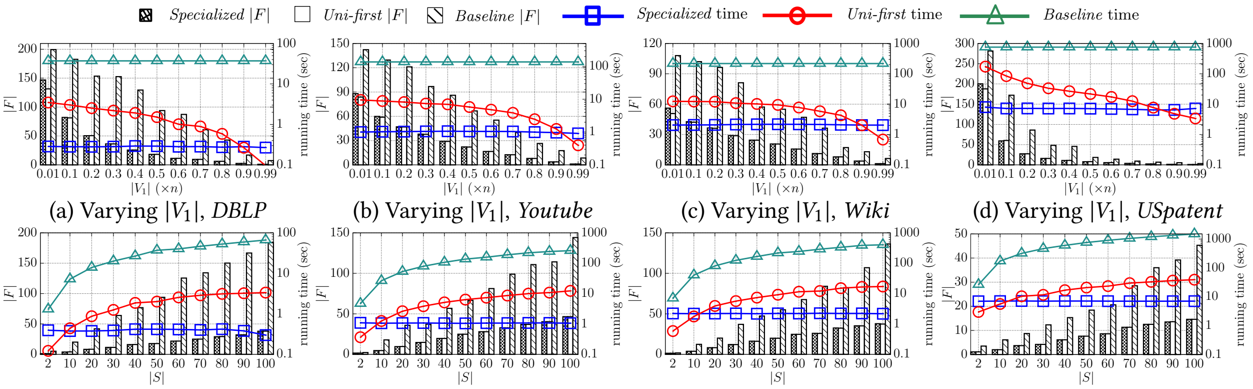
[6] All of them can be downloaded at http://konect.uni-koblenz.de.



(a) Varying $|V_1|$, *DBLP* (b) Varying $|V_1|$, *Youtube* (c) Varying $|V_1|$, *Wiki* (d) Varying $|V_1|$, *USpatent*

(e) Varying τ, *DBLP* (f) Varying τ, *Youtube* (g) Varying τ, *Wiki* (h) Varying τ, *USpatent*

Figure 5: *Specialized*, *Uni-first*, and *Baseline* on MVA-shortest-distance



(a) Varying $|V_1|$, *DBLP* (b) Varying $|V_1|$, *Youtube* (c) Varying $|V_1|$, *Wiki* (d) Varying $|V_1|$, *USpatent*

(e) Varying $|S|$, *DBLP* (f) Varying $|S|$, *Youtube* (g) Varying $|S|$, *Wiki* (h) Varying $|S|$, *USpatent*

Figure 6: *Specialized*, *Uni-first*, and *Baseline* on MVA-connectivity

approximation ratio 1.39, and returns the steiner tree as a solution to MVA-connecitivity. For *MVA-influence-minseed*, *Baseline* is the influence maximization algorithm in [15]. It doubles the parameter k of influence maximization (see Section 2) until the subset U found has $spread(U) \geq \tau$; this U is returned as a solution to MVA-influence-minseed. For MVA-reachability, *Baseline* first obtains the set $S \subseteq V$ of vertices reachable from the source vertex s (ignoring colors). It initializes U to the set of vertices reachable from s via 1-paths, and then repeats the following step until $|U| \geq \tau$: randomly pick a vertex u from $S \setminus U$ and add to U all the vertices on the shortest path from s to u (ignoring colors and assigning length 1 to every edge).

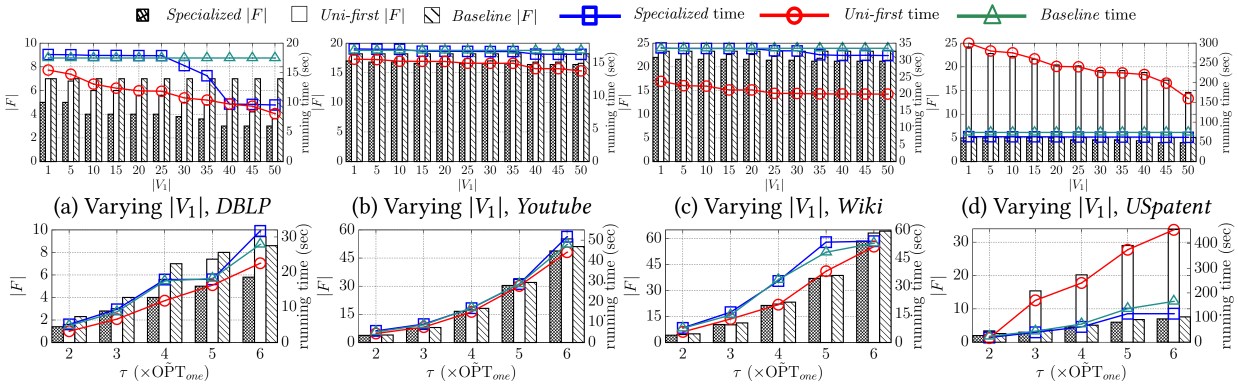
Environments. The experimentation was carried out on a machine equipped with an Intel 2.3 GHz Dural-Core CPU and 8 GB memory running MacOS 10.15.

On finding one solution. The first set of experiments aims to demonstrate (i) the MVA solution sizes under various parameter settings and (ii) the characteristics of *Specialized* and *Uni-first*.

Let us start with MVA-shortest-distance. In each experiment, the vertices in V_1 were picked uniformly at random (subject to a designated size $|V_1|$), so were the source and destination vertices s and t. Each result reported below is the average of at least 20 independent runs (a policy enforced throughout the experiments unless otherwise stated). The first experiment compared *Specialized*, *Uni-first*, and *Baseline* by varying $|V_1|$ and keeping the parameter τ at $1000/m$. Note that the value of τ is commensurate with the edge weights, which (by the generation process explained earlier) can be interpreted as the probability of crossing an edge after a random walk has stabilized.

Figures 5a-d present the solution size $|F|$ (columns, left y-axis) and the running time (curves, right y-axis) as a function of $|V_1|$. *Specialized* and *Baseline* returned the smallest solutions because they are optimal on this problem. *Specialized* did not exhibit an obvious pattern in running time. To see why, note that while a greater $|V_1|$ yields a smaller solution size k^*, it also increases the number of nodes to be processed; the behavior of *Specialized* resulted from the interplay of these two factors. The running time of *Uni-first* escalated with $|V_1|$ because its *next* function computes the shortest-path distances from s to all vertices in V_1. We omitted the running time of *Baseline* in Figures 5a-c because it ran longer than 500 seconds (i.e., over 100 times worse than our solutions).

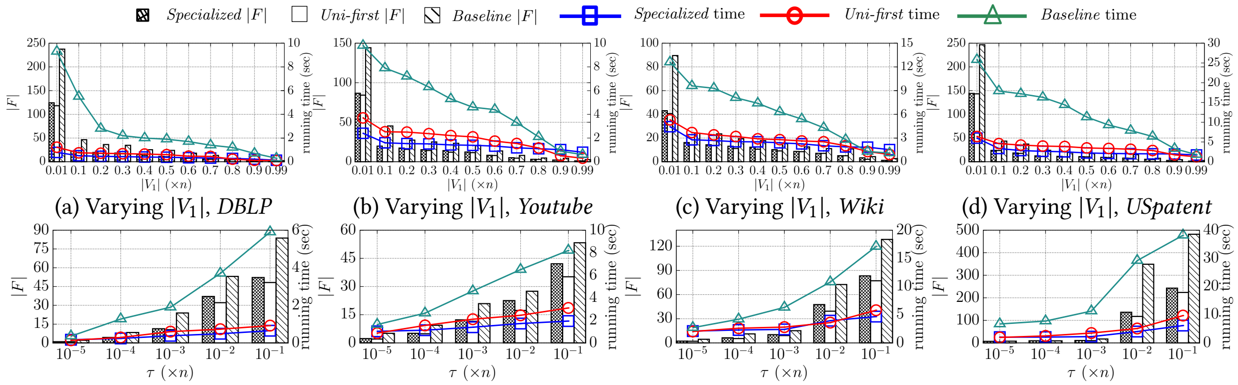
In the second experiment, we fixed $|V_1|$ to $0.5n$ while increasing τ. The results are presented in Figures 5e-h. All algorithms produced solutions of nearly the same size. *Specialized*'s running time did not follow an obvious pattern, but due to reasons different from Figures 5a-d. As τ increases, two factors create opposite effects. On the one hand, the value of k^* goes down (the predicate π becomes less stringent), which helps reduce time (the complexity depends on k^*; see Lemma 4.2). On the other hand, a greater τ forces *Specialized* to examine a larger portion of G (to check longer paths that may contain fewer 0-vertices), which induces higher cost. In contrast, *Uni-first* becomes faster for larger τ (which leads to smaller k^*) because its cost is proportional to k^*. Figures 5e-g again omitted *Baseline*'s running time, which was over 500 seconds.



(a) Varying $|V_1|$, *DBLP* (b) Varying $|V_1|$, *Youtube* (c) Varying $|V_1|$, *Wiki* (d) Varying $|V_1|$, *USpatent*

(e) Varying τ, *DBLP* (f) Varying τ, *Youtube* (g) Varying τ, *Wiki* (h) Varying τ, *USpatent*

Figure 7: *Specialized*, *Uni-first*, and *Baseline* on MVA-influence-minseed



(a) Varying $|V_1|$, *DBLP* (b) Varying $|V_1|$, *Youtube* (c) Varying $|V_1|$, *Wiki* (d) Varying $|V_1|$, *USpatent*

(e) Varying τ, *DBLP* (f) Varying τ, *Youtube* (g) Varying τ, *Wiki* (h) Varying τ, *USpatent*

Figure 8: *Specialized*, *Uni-first*, and *Baseline* on MVA-reachability

We then turned to MVA-connectivity. In each experiment, the set S of terminal vertices equaled any of the size-$|S|$ subsets of V_1 with the same probability. In Figures 6a-d, we set $|S| = 50$ and measured the performance of *Specialized*, *Uni-first*, and *Baseline* as a function of $|V_1|$; in Figures 6e-h, we fixed $|V_1| = 0.5n$ and measured their performance as a function of $|S|$. The solutions from *Uni-first* were nearly as small as those of *Specialized* (recall that the latter is protected by an approximation guarantee). *Specialized* had very stable efficiency due to its linear time complexity, while the time of *Uni-first* increased linearly with $|F|$. *Baseline* was considerably worse in both solution quality and efficiency in most settings.

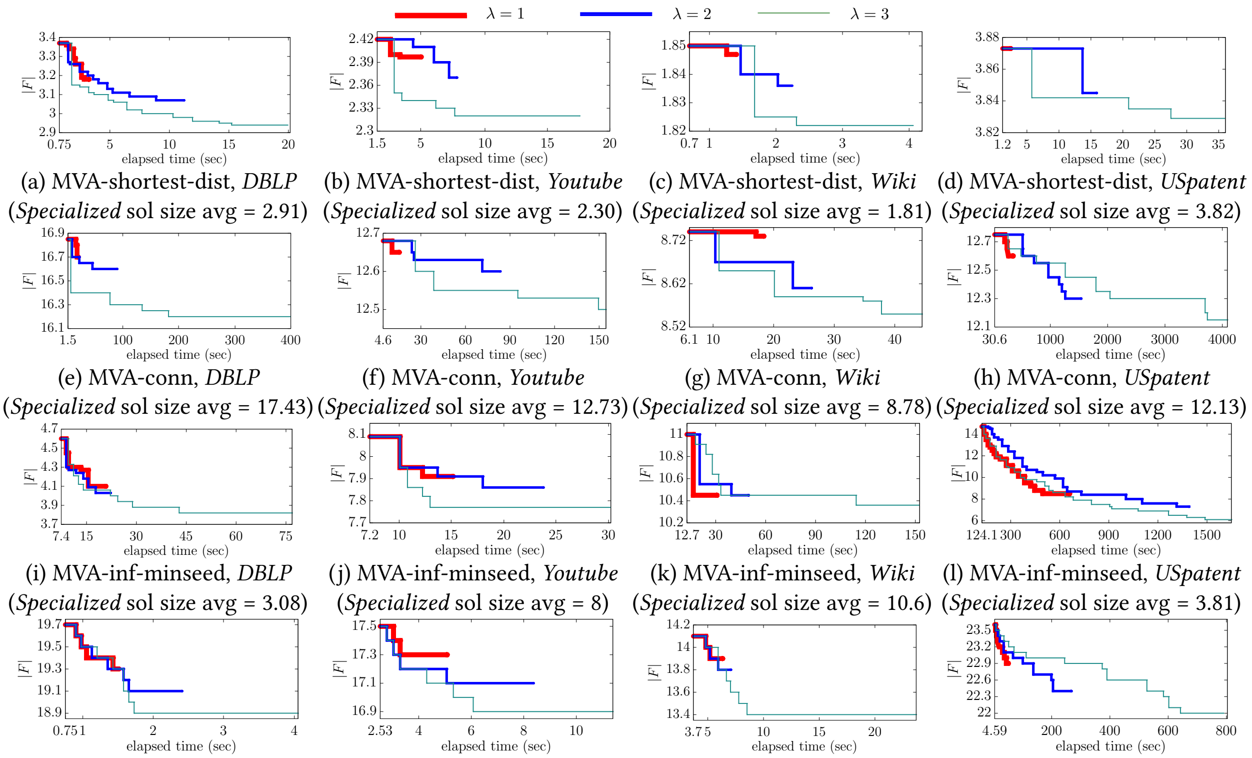
The experiments on MVA-influence-minseed had a similar design. Recall from Section 6 that *Specialized* starts by computing $\tilde{\text{OPT}}_{one}$ (roughly the spread of the most influential vertex in G). We varied the parameter τ from $2\tilde{\text{OPT}}_{one}$ to $6\tilde{\text{OPT}}_{one}$ for two considerations. First, this essentially requires the returned solution to have a spread *several times* that of the most influential vertex. Second, such τ led to solutions with *dozens* of vertices, a common practice followed by the influence-maximization literature (see, e.g., [1, 8, 9, 13, 38, 41]). We set the parameter $|\mathcal{R}_{next}|$ of *Uni-first* to 10000. The parameter controls the quality and efficiency of the *next* function (Section 8.2). Ideally, its value should vary according to the graph size, but this requires careful tuning. Fixing $|\mathcal{R}_{next}|$, as we did, means that the parameter tends to be small for large graphs. We chose this design to demonstrate (i) the benefits of *Specialized* (which has no such issues) and (ii) the self-recovery behavior of *Universal* (in the next set of experiments).

Figures 7a-d compares *Specialized*, *Uni-first*, and *Baseline* under different $|V_1|$ but the same $\tau = 4\tilde{\text{OPT}}_{one}$; Figures 7e-h shows another comparison under different τ but the same $|V_1| = 25$. On larger graphs, *Specialized* was noticeably superior to its competitors in solution size. Regarding efficiency, *Uni-first* outperformed *Specialized* on three graphs, while *Baseline* was consistently slower than *Specialized* in all cases.

Let us now attend to MVA-reachability. Figures 8a-d compare *Specialized*, *Uni-first*, *Baseline* under different $|V_1|$ but the same $\tau = 0.001n$, while Figures 8e-h give the comparison under different τ but the same $|V_1| = 0.5n$. Both *Specialized* and *Uni-first* had very similar performance. In retrospect, this should not be surprising because *Uni-first* can be regarded as a practical version of *Specialized*. They significantly outperformed *Baseline* in all settings.

Self-improving behavior of *Universal*. The second set of experiments aims to demonstrate what happens when *Universal* explores the polynomial core (Section 8.1), in particular, how additional processing time helps to improve the solution quality.

For a more meaningful understanding of "elapsed time", we use the cost of *Uni-first* as a reference. To explain, suppose that *Universal* finds the first solution in C_{first} seconds. We continue running the algorithm and measure, for each $i \geq 1$, the solution size $|F_i|$ at the moment when its running time reaches $(1 + \frac{i}{10}) \cdot C_{first}$ seconds. Doing so till termination will produce a non-ascending sequence:



(a) MVA-shortest-dist, *DBLP* (*Specialized* sol size avg = 2.91)

(b) MVA-shortest-dist, *Youtube* (*Specialized* sol size avg = 2.30)

(c) MVA-shortest-dist, *Wiki* (*Specialized* sol size avg = 1.81)

(d) MVA-shortest-dist, *USpatent* (*Specialized* sol size avg = 3.82)

(e) MVA-conn, *DBLP* (*Specialized* sol size avg = 17.43)

(f) MVA-conn, *Youtube* (*Specialized* sol size avg = 12.73)

(g) MVA-conn, *Wiki* (*Specialized* sol size avg = 8.78)

(h) MVA-conn, *USpatent* (*Specialized* sol size avg = 12.13)

(i) MVA-inf-minseed, *DBLP* (*Specialized* sol size avg = 3.08)

(j) MVA-inf-minseed, *Youtube* (*Specialized* sol size avg = 8)

(k) MVA-inf-minseed, *Wiki* (*Specialized* sol size avg = 10.6)

(l) MVA-inf-minseed, *USpatent* (*Specialized* sol size avg = 3.81)

(m) MVA-reachability, *DBLP* (*Specialized* sol size avg = 22.4)

(n) MVA-reachability, *Youtube* (*Specialized* sol size avg = 19.8)

(o) MVA-reachability, *Wiki* (*Specialized* sol size avg = 16.2)

(p) MVA-reachability, *USpatent* (*Specialized* sol size avg = 23.9)

Figure 9: Self-improvement of *Universal*

$|F_1|, |F_2|, |F_3|, \ldots$. Given an input and some fixed parameters, we launch 20 independent runs and average their sequences into an *aggregated* sequence Σ as follows. First, set I_{max} to the length of the 5th longest sequence (of the 20 runs). Then, generate an Σ of length I_{max} where $\Sigma[i]$ $(1 \le i \le I_{max})$ is the mean $|F_i|$ of all the runs. The Σ thus obtained manifests the self-improving ability of *Universal*. Also, denote by $\overline{C_{first}}$ the 20 runs' average C_{first} value.

Focusing on MVA-shortest-distance with $|V_1| = 0.1n$ and $\tau = 100/m$, Figures 9a-d plot the aggregated sequences for $\lambda = 1, 2$ and 3 on the four data graphs, respectively. Recall that λ is the parameter controlling the polynomial core (Lemma 8.1). In each diagram, the x-axis is the elapsed time, with the first value equal to $\overline{C_{first}}$; the y-axis is the solution size. For each aggregated sequence Σ, its $\Sigma[i]$ $(i \ge 1)$ is the y-value of the corresponding curve at $x = (1 + \frac{i}{10}) \cdot \overline{C_{first}}$. Note that the three curves of each diagram start from the same y-value, which is the size of the first solution (i.e., the output of *Uni-first*). The average solution size of *Specialized* is indicated under each figure. In the same fashion, Figures 9e-h present the results for MVA-connectivity with $|V_1| = 0.1n$ and $|S| = 10$, Figures 9i-l present the results for MVA-influence-minseed with $|V_1| = 5$ and $\tau = 3\tilde{\text{OPT}}_{one}$, and Figures 9m-p present the results for MVA-reachability with $|V_1| = 0.1n$ and $\tau = 0.001n$.

A higher λ defines a larger polynomial core and enables *Universal* to discover better solutions. For MVA-shortest-distance, the solutions of *Universal* at $\lambda = 3$ were almost as small as the optimal solutions found by *Specialized*. For MVA-connectivity and MVA-reachability, the $\lambda = 3$ solutions were often smaller than those of *Specialized*. The most substantial improvement occurred on MVA-influence-minseed, where *Universal* improved *Uni-first* by a substantial factor, especially on large graphs. As explained earlier, our choice of $|\mathcal{R}_{next}|$ (i.e., fixed, regardless of the graph size) implies a less powerful *next* function for more sizable graphs. The phenomenon suggests that our polynomial-core technique has good "correctional power", a nice property especially when the user supplies a mediocre *next* function. For practical use, we recommend choosing λ incrementally: start from $\lambda = 1$ and increase it if extra computation can be afforded. Based on our experiments, there would seldom be a need to go beyond $\lambda = 3$.

10 CONCLUSIONS AND FUTURE WORK

In real-world applications, the vertices in a data graph are often divided into two states: bad vs. normal. This paper introduces minimum vertex augmentation (MVA), whose goal is to fix the smallest number of bad vertices to fulfill a user-defined objective. MVA not only defines new problems but also motivates us to look at classical graph problems from a fresh perspective. We have proposed a set of techniques to design MVA algorithms with attractive performance guarantees, and developed a generic paradigm that a user can instantiate to deal with ad-hoc MVA problems. We have also presented extensive experimental results to demonstrate our techniques' effectiveness and efficiency on real data.

It is an exciting future direction to discover more generic techniques for designing MVA algorithms with excellent guarantees. For that purpose, we would like to invite the community to investigate challenging MVA problems beyond this article. Some examples include: flip the least number of 0-vertices such that the subgraph induced by 1-vertices (i) is bi-connected [40], (ii) has at least 10 paths from s to t with distance at most τ [44], (iii) contains at least τ occurrences of a user-defined pattern graph [36], (iv) can be converted to a pattern graph using matrix operations [6, 11], etc.

APPENDIX

A PROOF OF LEMMA 4.1

We will discuss only $color(v) = 1$ because the other case is similar. The direction $spdist(v|k) \le \min_{u \in \mathrm{IN}(v)} spdist(u|k) + w(u,v)$ is trivial. Next, we will show $spdist(v|k) \ge \min_{u \in \mathrm{IN}(v)} spdist(u|k) + w(u,v)$. Let P be a k-zero path from s to v with length $spdist(v|k)$. Consider the vertex u immediately before v in P; define P' as the prefix of P from s to u. Since P' uses no more than k 0-vertices, its length is at least $spdist(u|k)$. Hence, P has length at least $spdist(u|k) + w(u,v)$.

B PROOF OF LEMMA 4.2

Lemma B.1. *Every time Line 3 of* **MVA-shortest-distance** *removes $dist(v|k)$ from Q, it must hold that $dist(v|k) = spdist(v|k)$.*

Proof. We prove the lemma by induction. The first value removed is $dist(s|0) = 0$, on which the lemma obviously holds. Assuming correctness for all the values removed (by Line 3) so far, we will show that the lemma is also correct on the next removal.

Let $dist(v|k)$ be the value just removed from Q. We must have $spdist(v|k) \le dist(v|k)$.[7] Consider an arbitrary k-zero path P from s to v with length $spdist(v|k)$. For each vertex u on P, let k_u be the number of 0-vertices on the prefix of P from s to u. That prefix must have length $spdist(u|k_u)$.[8] Clearly, $k_u \le k$ and $spdist(u|k_u) \le spdist(v|k) \le dist(v|k)$.

We claim: *for every u on P other than v, $dist(u|k_u)$ has been removed from Q.* To prove the claim, define $\overline{u}$ as the last vertex on P such that $dist(\overline{u}|k_{\overline{u}})$ has been removed[9]. If the claim is wrong, $\overline{u}$ must be succeeded by a vertex $\overline{v}$ in P that is different from v. By the inductive assumption, $dist(\overline{u}|k_{\overline{u}}) = spdist(\overline{u}|k_{\overline{u}})$ when removed. Our algorithm then invoked **relax-edge** on $(\overline{u}, \overline{v}, k_{\overline{u}})$, leaving

$$\begin{array}{rcl} dist(\overline{v}|k_{\overline{v}}) & \le & dist(\overline{u}|k_{\overline{u}}) + w(\overline{u}, \overline{v}) \\ & = & spdist(\overline{u}|k_{\overline{u}}) + w(\overline{u}, \overline{v}) = spdist(\overline{v}|k_{\overline{v}}). \end{array}$$

This implies $dist(\overline{v}|k_{\overline{v}}) \trianglelefteq dist(v|k)$, contradicting the fact that $dist(v|k)$ was chosen for removal from Q.

To prove Lemma B.1, we fix u to be the vertex in P right before v. By the above claim, $dist(u|k_u)$ has been removed; when that happened, $dist(u|k_u) = spdist(u|k_u)$ (inductive assumption). The removal triggered **relax-edge** to be called on (u, v, k_u), leaving $dist(v|k) \le spdist(u|k_u) + w(u,v) = spdist(v|k)$. It thus follows that $dist(v|k) = spdist(v|k)$. □

Lemma B.2. *The sequence of values removed by Line 3 of* **MVA-shortest-distance** *is in ascending order under $\trianglelefteq$.*

Proof. Assume, for contradiction, that $dist(v|k)$ is removed before $dist(v'|k')$, but $dist(v'|k') \trianglelefteq dist(v|k)$. Clearly, $v \neq s$. Let P' be an arbitrary k'-zero path from s to v' with length $dist(v'|k') = spdist(v'|k')$ (Lemma B.1). For each u on P', define k_u as the number of 0-vertices on the prefix of P' from s to u. Let $\overline{u}$ be the last vertex on P' such that $dist(\overline{u}|k_{\overline{u}})$ is removed before $dist(v|k)$; $\overline{u}$ definitely exists because $dist(s|0)$ is the first removed and $v \neq s$.

Let $\overline{v}$ be the successor of $\overline{u}$ on P'. After removing $dist(\overline{u}|k_{\overline{u}})$, our algorithm invoked **relax-edge** on $(\overline{u}, \overline{v}, k_{\overline{u}})$, after which

$$\begin{array}{rcl} dist(\overline{v}|k_{\overline{v}}) & \le & dist(\overline{u}|k_{\overline{u}}) + w(\overline{u}, \overline{v}) \\ \text{(Lemma B.1)} & = & spdist(\overline{u}|k_{\overline{u}}) + w(\overline{u}, \overline{v}) \\ & = & spdist(\overline{v}|k_{\overline{v}}) \le spdist(v'|k') \end{array}$$

where the equality holds through only if $\overline{v} = v'$. This, together with the obvious fact that $k_{\overline{v}} \le k'$, implies that either $\overline{v} = v'$ or $dist(\overline{v}|k_{\overline{v}}) \trianglelefteq dist(v'|k')$. In both cases, we must have by transitivity that $dist(\overline{v}|k_{\overline{v}}) \trianglelefteq dist(v|k)$, which contradicts the algorithm's choice to remove $dist(v|k)$ before $dist(\overline{v}|k_{\overline{v}})$. □

Lemma B.3. *Line 3 of* **MVA-shortest-distance** *eventually removes a value $dist(u|k)$ with $u = t$. Furthermore, when this happens, $k = k^*$, where k^* is as defined in Lemma 4.2.*

Proof. Assume, on the contrary, that Line 3 never managed to remove $dist(t|k^*)$ from Q. Let P be an arbitrary k^*-zero path from s to t with length $spdist(t|k^*)$. For each u on P, define k_u as the number of 0-vertices on the prefix of P from s to u. Let $\overline{u}$ be the last vertex on P such that $dist(\overline{u}|k_{\overline{u}})$ had been removed by Line 3 at the algorithm's termination. As $\overline{u} \neq t$, $\overline{u}$ must have a successor $\overline{v}$ on P.

After removing $dist(\overline{u}|k_{\overline{u}})$, the algorithm invoked **relax-edge** on $(\overline{u}, \overline{v}, k_{\overline{u}})$, after which

$$dist(\overline{v}|k_{\overline{v}}) = spdist(\overline{v}|k_{\overline{v}}) \le spdist(t|k^*) \le \tau$$

where the first equality is due to Lemma B.1. The algorithm terminated while having $dist(\overline{v}|k_{\overline{v}})$ in Q. This implies that termination could only have happened at Line 4, indicating that Line 3 must have removed a value $dist(t|k')$ for some $k' \neq k^*$. The rest of the argument proceeds as follows:

- If $k' < k^*$, then $dist(t|k') = spdist(t|k') > \tau$ by definition of k^*. Hence, $dist(t|k')$ is inactive, and thus must rank after $dist(\overline{v}|k_{\overline{v}})$ under $\trianglelefteq$, contradicting the algorithm's choice to remove $dist(t|k')$ from Q.
- If $k' > k^*$, then by $k_{\overline{v}} \le k^* < k'$ we know that $dist(\overline{v}|k_{\overline{v}}) \trianglelefteq dist(t|k')$, again contradicting the fact that the algorithm chose to remove $dist(t|k')$ from Q. □

It remains to prove the running time. By Lemmas B.2 and B.3, the values removed at Line 3 must rank no later than $dist(t|k^*)$ under $\trianglelefteq$, and that every such value can be removed at most once. Hence, the total number of calls to **relax-edge** is at most $k^* m$. With Q implemented as with a binary search tree, every call takes $\tilde{O}(1)$ time. The total cost of all those calls is bounded by $\tilde{O}(k^* m)$.

C PROOF OF LEMMA 6.1

We will prove the lemma by assuming that v_{first} is a vertex in an optimal solution to the MVA-influence-minseed problem. If not, it is straightforward to adapt the subsequent argument by accounting for this single vertex.

[7] $dist(v|k)$ is the length of one possible k-zero path from s to v, whereas $spdist(v|k)$ is the minimum length of all such paths.

[8] Otherwise we can replace the prefix with a shorter k_u-zero path from s to u, yielding a shorter k-zero path from s to v, contradicting the definition of P.

[9] $\overline{u}$ definitely exists because $dist(s|0)$ must have been removed.

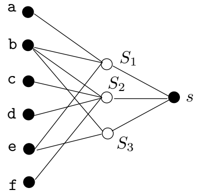


Figure 10: Reduction illustration

We tackle MVA-influence-minseed as follows:

```
1. $k \leftarrow 1$
2. while $k \le n$ do
3.     $(U_k, X_k) \leftarrow$ a 0.5-approx IMP solution under para. $k$
4.     if $X_k \ge \tau/4$ then return $U_k$
5.     else double $k$
6. return $V$
```

We will discuss only the event where all the IMP solutions at Line 3 are correct; as there are $O(\log n)$ iterations, the event happens with probability at least $1 - \frac{\log n}{n^3} = 1 - O(1/n^2)$.

Let k' be the value of k at termination. It is clear that

$$spread(U_{k'} \cup V_1) \ge \frac{X_k}{1.5} \ge \frac{\tau/4}{1.5} = \Omega(\tau).$$

Next we prove $k' \le 1 + 2k^*$. The claim obviously holds if $k' = 1$. Now consider $k' > 1$. Since the algorithm does not terminate at $k = k'/2$, we have $X_{k'/2} < \tau/4$. Let $U^*_{k'/2}$ be an optimal solution to IMP under parameter $k = k'/2$. It must hold that

$$spread(U^*_{k'/2} \cup V_1) \le \frac{spread(U_{k'/2} \cup V_1)}{0.5} \le \frac{X_{k'/2}}{0.25} < \tau.$$

Therefore, $k^* > k'/2$ by definition of k^*.

D NP-HARDNESS OF MVA-REACHABILITY

We will reduce the *set cover* problem to MVA-reachability. In the former problem (which is well-known to be NP-hard), the input is a collection $\mathcal{S}$ of sets $S_1, S_2, ..., S_m$ ($m \ge 1$). Define $U = \bigcup_{i=1}^{m} S_i$. A $\mathcal{T} \subseteq \mathcal{S}$ is a *covering sub-collection* if the union of the sets in $\mathcal{T}$ equals U. The goal is to return a covering sub-collection $\mathcal{T}$ of the minimum $|\mathcal{T}|$.

Construct an undirected graph $G = (V, E)$ as follows. V has $|U| + m + 1$ vertices:

- for each element in U, V has a corresponding vertex, which we will refer to as an *element vertex*;
- for each set $S_i \in \mathcal{S}$ ($i \in [1, m]$), V has a corresponding vertex, which we will refer to as a *set vertex*;
- in addition, V has a *special vertex* s.

Color all the set vertices 0 and the other vertices 1. The edges in E are decided as follows:

- for each element $u \in U$ and each set $S_i \in \mathcal{S}$, add an edge connecting the element vertex u and the set vertex S_i if $u \in S_i$;
- for each set $S_i \in S$, add an edge between the special vertex s and the set vertex S_i.

Figure 10 shows an example for $\mathcal{S} = \{S_1, S_2, S_3\}$ where $S_1 = \{\mathsf{a}, \mathsf{b}, \mathsf{e}\}$, $S_2 = \{\mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{f}\}$, and $S_3 = \{\mathsf{b}, \mathsf{e}\}$.

Every integer $\tau \in [|U| + 1, |V|]$ defines an instance of MVA-reachability on G. For each τ, solve its instance optimally, obtaining a set F_τ of vertices (note that F_τ contains only set vertices). If s can reach *all* the element vertices after flipping the vertices in F_τ, call F_τ *legal*. Define τ^* as the smallest τ giving a legal F_τ. For the original set cover problem, we add to $\mathcal{T}$ the set $S_i \in \mathcal{S}$ if and only if F_{τ^*} contains the set vertex S_i, and return the $\mathcal{T}$ thus obtained.

It is obvious that the union of the sets in $\mathcal{T}$ equals U. Next, we argue that $\mathcal{T}$ has the smallest size among all the covering sub-collections.

Suppose that $\mathcal{T}^*$ is an optimal covering sub-collection and that $|\mathcal{T}^*| < |\mathcal{T}|$. Define $F^* = \{S_i \in \mathcal{S} \mid S_i \in \mathcal{T}^*\}$. Consider the MVA-reachability instance for $\tau^* = |U| + 1 + |\mathcal{T}^*|$. The definition of $\mathcal{T}^*$ implies that F^* is a solution to that instance. Consider the solution F_{τ^*} found by our algorithm on this instance; the optimality of F_{τ^*} ensures $|F_{\tau^*}| \le |F^*| = |\mathcal{T}^*|$. Note that s cannot reach all the element vertices after flipping the vertices in F_{τ^*}; otherwise, our algorithm would have returned a covering sub-collection with size $|F_{\tau^*}| \le |\mathcal{T}^*| < |\mathcal{T}|$. Remember, however, that s must reach at least τ^* 1-vertices (including s itself) after flipping the vertices in F_{τ^*}. This means that at least $\tau^* - |U| = |\mathcal{T}^*| + 1$ of those 1-vertices must be the set vertices in F_{τ^*}. Therefore, $|F_{\tau^*}| \ge |\mathcal{T}^*| + 1$, giving a contradiction.

E PROOF OF LEMMA 8.1

We consider an alternative question. Let $\mathscr{T}$ be a perfect binary tree. Mark a node z in $\mathscr{T}$ if the root-to-z path in $\mathscr{T}$ makes at most k_{first} left turns (Facts 1 and 2) and at most λ right turns (bounded-death heuristic). The question is: what is the number of nodes marked? Next, we will show that the answer is no more than $(k_{first} + 1)^{\lambda+1}$, which will establish the lemma.

A simple inductive argument suffices. The claim is obviously correct for $\lambda = 0$. Assuming correctness for $\lambda = c$, next we prove the claim for $\lambda = c + 1$. Let z_{first} be the lowest marked node on the leftmost path of $\mathscr{T}$. Denote by P the path from the root to z_{first}. Consider an arbitrary node z on P; let z' be the right child of z. By the inductive argument, the number of marked nodes in the subtree of z' is at most $(1 + k_{first})^c$. The claim thus follows from the fact that P has $k_{first} + 1$ nodes.

REFERENCES

[1] Akhil Arora, Sainyam Galhotra, and Sayan Ranu. 2017. Debunking the Myths of Influence Maximization: An In-Depth Benchmarking Study. In *Proceedings of ACM Management of Data (SIGMOD)*. 651–666.

[2] Piotr Berman and Viswanathan Ramaiyer. 1994. Improved Approximations for the Steiner Tree Problem. *Journal of Algorithms* 17, 3 (1994), 381–408.

[3] Stephan Beyer and Markus Chimani. 2014. Steiner Tree 1.39-Approximation in Practice. In *MEMICS*, Vol. 8934. 60–72.

[4] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. 2001. Towards Sensor Database Systems, Vol. 1987. 3–14.

[5] Christian Borgs, Michael Brautbar, Jennifer T. Chayes, and Brendan Lucier. 2014. Maximizing Social Influence in Nearly Optimal Time. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 946–957.

[6] Robert Brijder, Floris Geerts, Jan Van den Bussche, and Timmy Weerwag. 2018. On the Expressive Power of Query Languages for Matrices. In *Proceedings of International Conference on Database Theory (ICDT)*. 10:1–10:17.

[7] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. 2013. Steiner Tree Approximation via Iterative Randomized Rounding. *Journal of the ACM (JACM)* 60, 1 (2013), 6:1–6:33.

[8] Wei Chen, Chi Wang, and Yajun Wang. 2010. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of ACM Knowledge Discovery and Data Mining (SIGKDD)*. 1029–1038.

[9] Sainyam Galhotra, Akhil Arora, and Shourya Roy. 2016. Holistic Influence Maximization: Combining Scalability and Efficiency with Opinion-Aware Models. In *Proceedings of ACM Management of Data (SIGMOD)*. 743–758.

[10] Michael R Garey and David S Johnson. 1979. *Computers and intractability*. Vol. 174. Freeman San Francisco. 208–209 pages.

[11] Floris Geerts. 2020. When Can Matrix Query Languages Discern Matrices?. In *Proceedings of International Conference on Database Theory (ICDT)*. 12:1–12:18.

[12] Michel X. Goemans, Neil Olver, Thomas Rothvoß, and Rico Zenklusen. 2012. Matroids and integrality gaps for hypergraphic steiner tree relaxations. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*. 1161–1176.

[13] Amit Goyal, Francesco Bonchi, and Laks V. S. Lakshmanan. 2011. A Data-Based Approach to Social Influence Maximization. *Proceedings of the VLDB Endowment (PVLDB)* 5, 1 (2011), 73–84.

[14] Amit Goyal, Francesco Bonchi, Laks V. S. Lakshmanan, and Suresh Venkatasubramanian. 2013. On minimizing budget and time in influence propagation over social networks. *Social Netw. Analys. Mining* 3, 2 (2013), 179–192.

[15] Qintian Guo, Sibo Wang, Zhewei Wei, and Ming Chen. 2020. Influence Maximization Revisited: Efficient Reverse Reachable Set Generation with Bound Tightened. In *Proceedings of ACM Management of Data (SIGMOD)*. 2167–2181.

[16] Jurriaan Hage, Tero Harju, and Emo Welzl. 2003. Euler Graphs, Triangle-Free Graphs and Bipartite Graphs in Switching Classes. *Fundam. Inform.* 58, 1 (2003), 23–37.

[17] Ryan B. Hayward. 1996. RecognizingP_3-Structure: A Switching Approach. *J. Comb. Theory, Ser. B* 66, 2 (1996), 247–262.

[18] Stefan Hougardy and Hans Jurgen Promel. 1999. A 1.598 Approximation Algorithm for the Steiner Problem in Graphs. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 448–453.

[19] Eva Jelínková and Jan Kratochvíl. 2014. On Switching to H-Free Graphs. *Journal of Graph Theory* 75, 4 (2014), 387–405.

[20] T.R. Jensen and B. Toft. 1995. *Graph Coloring Problems*. John Wiley & Sons.

[21] Kyomin Jung, Wooram Heo, and Wei Chen. 2012. IRIE: Scalable and Robust Influence Maximization in Social Networks. In *Proceedings of International Conference on Management of Data (ICDM)*. 918–923.

[22] Marek Karpinski and Alexander Zelikovsky. 1997. New Approximation Algorithms for the Steiner Tree Problems. *J. Comb. Optim.* 1, 1 (1997), 47–65.

[23] David Kempe, Jon M. Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *Proceedings of ACM Knowledge Discovery and Data Mining (SIGKDD)*. 137–146.

[24] Jon Lee. 2004. *A First Course in Combinatorial Optimization*. Cambridge University Press.

[25] Jong-Ryul Lee and Chin-Wan Chung. 2014. A fast approximation for influence maximization in large social networks. In *Proceedings of International World Wide Web Conferences (WWW)*. 1157–1162.

[26] R.M. R. Lewis. 2015. *A Guide to Graph Colouring: Algorithms and Applications*. Springer.

[27] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. 2005. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems (TODS)* 30, 1 (2005), 122–173.

[28] Kurt Mehlhorn. 1988. A Faster Approximation Algorithm for the Steiner Problem in Graphs. *Information Processing Letters (IPL)* 27, 3 (1988), 125–128.

[29] Hung T. Nguyen, My T. Thai, and Thang N. Dinh. 2016. Stop-and-Stare: Optimal Sampling Algorithms for Viral Marketing in Billion-scale Networks. In *Proceedings of ACM Management of Data (SIGMOD)*. 695–710.

[30] Naoto Ohsaka, Takuya Akiba, Yuichi Yoshida, and Ken-ichi Kawarabayashi. 2014. Fast and Accurate Influence Maximization on Large Networks with Pruned Monte-Carlo Simulations. 138–144.

[31] Ho Lam Pang and Leizhen Cai. 2019. Complexity of Vertex Switching on Edge-Bicolored Graphs. In *Proceedings of International Conference on Algorithms and Complexity (CIAC)*. 339–351.

[32] Hans Jürgen Prömel and Angelika Steger. 1997. RNC-Approximation Algorithms for the Steiner Problem. In *Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS)*. 559–570.

[33] Gabriel Robins and Alexander Zelikovsky. 2005. Tighter Bounds for Graph Steiner Tree Approximation. *SIAM J. Discret. Math.* 19, 1 (2005), 122–134.

[34] Alexander Schrijver. 2004. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer-Verlag.

[35] J. J. Seidel. 1991. A survey of two-graphs. *Geometry and Combinatorics* (1991), 146–176.

[36] Zhao Sun, Hongzhi Wang, Haixun Wang, Bin Shao, and Jianzhong Li. 2012. Efficient Subgraph Matching on Billion Node Graphs. *Proceedings of the VLDB Endowment (PVLDB)* 5, 9 (2012), 788–799.

[37] Jing Tang, Xueyan Tang, Xiaokui Xiao, and Junsong Yuan. 2018. Online Processing Algorithms for Influence Maximization. In *Proceedings of ACM Management of Data (SIGMOD)*. 991–1005.

[38] Youze Tang, Yanchen Shi, and Xiaokui Xiao. 2015. Influence Maximization in Near-Linear Time: A Martingale Approach. In *Proceedings of ACM Management of Data (SIGMOD)*. 1539–1554.

[39] Youze Tang, Xiaokui Xiao, and Yanchen Shi. 2014. Influence maximization: near-optimal time complexity meets practical efficiency. In *Proceedings of ACM Management of Data (SIGMOD)*. 75–86.

[40] Robert Endre Tarjan. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM Journal of Computing* 1, 2 (1972), 146–160.

[41] Chi Wang, Wei Chen, and Yajun Wang. 2012. Scalable influence maximization for independent cascade model in large-scale social networks. *Data Min. Knowl. Discov.* 25, 3 (2012), 545–576.

[42] Laurence A. Wolsey. 1982. An analysis of the greedy algorithm for the submodular set covering problem. *Comb.* 2, 4 (1982), 385–393.

[43] Yong Yao and Johannes Gehrke. 2003. Query Processing in Sensor Networks. In *Proceedings of Biennial Conference on Innovative Data Systems Research (CIDR)*.

[44] Ziqiang Yu, Xiaohui Yu, Nick Koudas, Yang Liu, Yifan Li, Yueting Chen, and Dingyu Yang. 2020. Distributed Processing of k Shortest Path Queries over Dynamic Road Networks. In *Proceedings of ACM Management of Data (SIGMOD)*. 665–679.

[45] Thomas Zaslavsky. 1982. Signed graphs. *Discret. Appl. Math.* 4, 1 (1982), 47–74.

[46] Alexander Zelikovsky. 1993. An 11/6-Approximation Algorithm for the Network Steiner Problem. *Algorithmica* 9, 5 (1993), 463–470.

[47] Jianwen Zhao and Yufei Tao. 2021. Minimum Vertex Augmentation. *Technical Report at http://www.cse.cuhk.edu.hk/∼taoyf/mva-full.pdf* (2021).