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

1 INTRODUCTION

Many graph-based applications endow vertices with binary states, such as reliable/unreliable (e.g., sensors), 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 sensor’s battery, influence a non-customer with advertising, and authenticating an unreliable output, and so on. 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, 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 \( \pi \). This is equivalent to augmenting \( V_1 \) by flipping the \( 0 \)-vertices to upload \( \pi \) on the subgraph induced by all the (old and new) \( 1 \)-vertices. With \( \pi \) formulated differently, the framework defines a class of problems that we call the MVA class and denote as \( \mathcal{V}_{\text{MVA}} \). Some problems in the class extend conventional graph problems in non-trivial manners, while certain MVA problems do not necessarily have “conventional” counterparts. The next subsection discusses a representative problem selection.

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 \( \pi \). In other words, the goal is to minimally augment the subset of \( 1 \)-vertices (people), verified/non-verified (intermediate outputs in a workflow), or traditional graph problems in non-trivial manners, while certain MVA 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.

PVLDB Reference Format:
doi:10.14778/3461535.3461536

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 \( \pi = " \text{the subgraph induced by } F \cup V_1 \text{ has a path from } s \text{ to } t \text{ whose total length (weight sum of all the edges on the path) is at most } \tau." \). 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 \( \pi = " \text{the subgraph induced by } F \cup V_1 \text{ has all the vertices of } S \text{ in the same connected component}." \). Consider the graph \( G \) in Figure 1b where \( V_1 = \{t_1, t_2, t_3, b, d\} \). For \( S = \{t_1, t_2, t_3\} \), the optimal solution is to flip \( F = \{a, e\} \), 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 \( d, f, \) 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 \( \text{spread}(S) \) the overall influence of \( S \) in this cascading process (the detailed calculation of \( \text{spread}(S) \) will appear in Section 2). In MVA-influence-minseed, the predicate \( \pi = \text{"spread}(F \cup V_1) \geq \tau" \) 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 \( V_1 = \{b, c, d, e\} \), however, \( j \) becomes better. The presence of \( b, c, d, \) and \( e \) “blocks” from \( f, g, h, i \) such that advertising to \( a \) will create no more impact on \( \{f, g, h, i\} \) 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 \( \pi = " s \text{ can reach at least } \tau \text{ vertices in the subgraph induced by } F \cup V_1." \). 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
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. Market maven, sales-person, 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 $G$. 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. Quickly how 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-reaching.

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 $\mathcal{C}_\text{MVA}$ 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 $\mathcal{C}_\text{MVA}$ 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 representative problems are selected to illustrate three techniques to approach MVA:

- **Dynamic programming:** The technique allows us to settle MVA-shortest-distance in $\tilde{O}(n + k^2m)$ time\(^{4}\) 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 $\pi$.

- **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 $\tilde{O}(n + k^*)$ time a solution $F$ with size $O(1 + k^*)$ and $\text{spread}(F \cup V_1) = \Omega(\tau)$.

4\(^{4}The notation $\tilde{O}(\cdot)$ hides a polylogarithmic to the problem size.
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 \( \{u, v\} \). Each vertex \( v \in V \) has a color in \( \{0, 1\} \), denoted as \( \text{color}(v) \). Define \( V_0 = \{v \in V \mid \text{color}(v) = 0\} \) and \( V_1 = \{v \in V \mid \text{color}(v) = 1\} \).

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

Definition 2.1 (Monotonicity). Function \( \pi \) is monotone if, for any \( U_1 \subseteq U_2 \subseteq V \), \( \pi(G_{U_1}) = \text{true} \) implies \( \pi(G_{U_2}) = \text{true} \).

Every such \( \pi \) defines an MVA-problem.

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

We will refer to \( F \) as a solution if \( \pi(G_{F \cup V_1}) = \text{true} \), 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 \( \pi(G_U) = \text{false} \) 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 \( \mathcal{F}_\text{MVA} \) is the set of all MVA-problems defined by monotone functions \( \pi \).

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 \( \pi(G_U) = \text{true} \) if and only if \( G_U \) contains a path from \( s \) to \( t \) whose length is at most \( \tau \) (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 \( \tau \), 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 \( \pi(G_U) = \text{true} \) 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 \( \pi(G_U) = \text{true} \) if and only if \( \text{spread}(U) \geq \tau \).

The definition of \( \text{spread}(U) \) 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 \( E_{\text{rand}} \) the set of edges retained and by \( G_{\text{rand}} = (V, E_{\text{rand}}) \) 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 \( G_{\text{rand}} \). The value of \( \text{spread}(U) \) is the expectation of the random variable \( X \).

Example. 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 \( G_{\text{rand}} \). For \( U = \{a, c\}, X = 6 \) because \( U \) can reach \( a, b, c, d, f \) and \( g \) in \( G_{\text{rand}} \).

The plain version of MVA-influence-minseed finds a subset \( U \subseteq V \) with the smallest size to satisfy \( \text{spread}(U) \geq \tau \). This is called minimum target set selection (MINTSS) [14] and dual to the influence maximization problem [23] (the latter returns a set \( U \subseteq V \) with the maximum \( \text{spread}(U) \) 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 \( \pi(G_U) = \text{true} \) if and only if \( s \) can reach at least \( \tau \) 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 \( \pi \).

\( \mathcal{F}_\text{MVA} \) 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]. Mehrotra [28] designed a practical 2-approximate algorithm that runs in \( O(n + m) \) 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
This section will settle MVA-shortest-distance with dynamic programming. The algorithm runs in time $O(k(m + n))$ 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 $O(k'(m + n))$ time\(^2\) to find a solution $U$ of size $O(k')$ with $\text{spread}(U) = \Omega(\tau)$. 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 $\tau$ vertices, and (iii) $s$ can reach all the vertices in $G'$. The problem can be settled in linear time by running DFS/BFS from $s$ and returning the first $\tau$ vertices seen.

\text{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 $O(n + m)$ 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 $e = (u, v)$, $w_0(u, v)$ 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 $\text{spdist}(v[k])$ as the minimum length of all $k$-zero paths $P$ from $s$ to $v$. We prove in Appendix A:

**Lemma 4.1.** $\text{spdist}(v[k])$ equals

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

where $\text{IN}(v)$ 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 $\text{dist}(v[k])$ values for different $v \in V$ and $k \geq 0$. The value of $\text{dist}(v[k])$ is at least $\text{spdist}(v[k])$ and decreases monotonically as the algorithm runs. A $\text{dist}(v[k])$ is active if it is at most $\tau$, or inactive otherwise.

We impose a total order $\preceq$ on $Q$: $\text{dist}(v[k]) \preceq \text{dist}(v'[k'])$ if

- (both active, $k = k'$) $\text{dist}(v[k])$ is active but $\text{dist}(v'[k'])$ is not;
- (both active) $k < k'$;
- (both active and $k = k'$) $\text{dist}(v[k]) < \text{dist}(v'[k'])$.

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

**Algorithm**

**Algorithm MVA-shortest-distance**

1. insert triplet $(\text{dist}(s[0]), s, 0)$ into $Q$ where $\text{dist}(s[0]) = 0$ is the key
2. repeat
3. remove from $Q$ the triplet $(\text{dist}(u[k]), u, k)$ with the smallest $\text{dist}(u[k])$ under $\preceq$
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 $\text{dist}(u[k]) + w(u, v) < \text{dist}(v[k])$ then
   - update/insert the triplet $(\text{dist}(v[k]), v, k)$ in $Q$ with
     - $\text{dist}(v[k]) \leftarrow \text{dist}(u[k]) + w(u, v)$
2. if color$(v) = 0$ and $\text{dist}(u[k]) + w(u, v) < \text{dist}(v[k] + 1)$ then
   - update/insert the triplet $(\text{dist}(v[k] + 1), v, k + 1)$ in $Q$ with
     - $\text{dist}(v[k] + 1) \leftarrow \text{dist}(u[k]) + w(u, v)$

**Example**

We illustrate the algorithm on the input graph in Figure 1A, setting $\tau = 4$ (recall that all edges have weight 1). Initially, $\text{dist}(s[0]) = 0$, whereas $\text{dist}(u[k]) = \infty$ for every vertex $u \neq s$ and integer $k \geq 1$. After removing $\text{dist}(s[0])$ from $Q$, we perform **relax-edge** on $(s, a, 0)$ and $(s, c, 0)$, after which $\text{dist}(a[1]) = 1$ and $\text{dist}(c[0]) = 1$ are added to $Q$. As $\text{dist}(c[0]) \preceq \text{dist}(a[1])$, the next entry removed from $Q$ is $\text{dist}(c[0])$, triggering **relax-edge** on $(c, d, 0)$, which adds $\text{dist}(d[1]) = 2$ to $Q$. Similarly, $\text{dist}(a[1])$ is then removed; and **relax-edge** on $(a, b, 1)$ adds $\text{dist}(b[2]) = 2$ to $Q$. The algorithm then removes $\text{dist}(d[1]) = 2$, performs **relax-edge** on $(d, e, 1)$, adds $\text{dist}(e[1]) = 3$, removes $\text{dist}(e[1]) = 3$, performs **relax-edge** on $(e, t, 1)$, adds $\text{dist}(t[1]) = 4$, removes $\text{dist}(t[1]) = 4$, 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 $\text{spdist}(t[k']) \leq \tau$. The execution time is in $O(n + k'm)$.

Recall that $k'$ is the size of an optimal solution. We still need to find the path $P$ from $s$ to $t$ corresponding to $\text{spdist}(t[k'])$; namely, $P$ has length $\text{spdist}(t[k'])$ and exactly $k'$ 0-vertices. After that, the $F$ returned is just the set of 0-vertices in $P$. To obtain $P$, we apply the standard "parent pointer trick" that converts a shortest-distance algorithm to a shortest-path one. Specifically, we maintain a pointer $\text{parent}(v[k])$ for each vertex $v$ and $k \geq 0$. Every time Line 2 (or 4) of **relax-edge** is executed, we set $\text{parent}(v[k])$ (or $\text{parent}(v[k] + 1)$, 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 non-trivial 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 $G^* \subseteq (V, E)$ with the same vertex and edge sets. Specifically, to every edge $e = [u, v]$, $w^*$...
we assign the weight (i) 1 if both and have color 0, (ii) 1/2 if only one of or has color 0, or (iii) 0 if neither does. To understand this in another way, consider processing each 0-vertex by injecting a weight of 1/2 to every incident edge. Thus, 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 in with respect to the terminal vertices in S;
2. set to the set of 0-vertices in T.

**Example.** Figure 3 shows the graph obtained from the input graph in Figure 1b. Assuming , the optimal steiner tree in contains vertices , , , , , , and .

We cannot execute Step (1) efficiently because the steiner tree problem is NP-hard. Assuming the availability of an -approximate steiner tree algorithm, we replace Steps (1) and (2) with an -approximate steiner tree in . The modified algorithm ensures:

**Lemma 5.1.** We return an of size at most \( k^* \cdot (1 + \frac{|S|}{2k^*}) \), 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 \( o(1 + |S|/2k^*) \). By adopting the 2-approximate steiner tree algorithm of [28] (Section 3), we achieve approximation ratio \( 2 + |S|/k^* \) and running time \( O(n + m) \).

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 planar version of MVA-connectivity).

**5.2 Proof of Lemma 5.1**

We start by proving a property of G*:

**Lemma 5.2.** Let be any tree embedded in G* (i.e., each edge of in G* such that all the leaf nodes of have color 1). Then, the weight of 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 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

\[
\sum_{(u, v) \in T} (w(u) + w(v)) = \sum_{u \in T} w(u) \cdot d_T(u)
\]

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, \( k^* = |F^*| \)) to the MVA-connectivity problem. \( G_{V_1 \cup F^*} \) must contain a steiner 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.

\[
C = \sum_{u \in F^*} \frac{d_T(u)}{2} - \sum_{u \in F^*} \left( \frac{d_T(u)}{2} - 1 \right) = |F^*| + \sum_{u \in F^*} (d_T(u)/2 - 1) = |F^*| + \sum_{u \in X} d_T(u) - |X| + \frac{1}{2} \sum_{u \in X} d_T(u) - 1 + 1 = |F^*| + \frac{1}{2} \sum_{u \in X} d_T(u) - 1 + 1 = \frac{1}{2} \sum_{u \in X} d_T(u) - 1 + 1.
\]

Thus, we can find in \( \tilde{O}(n + m) \) time with probability at least \( 1 - \frac{1}{2} \).

6 **GREEDY (MVA-INFLUENCE-MINSEED)**

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

- \( OPT_{one} = \max_{v \in V} spread(v) \);
- \( v_{first} \) as an arbitrary vertex with \( spread(v_{first}) \geq \frac{1}{2} OPT_{one} \);
- \( OPT_{one} \) as an arbitrary value in \( [\frac{1}{2} OPT_{one}, OPT_{one}] \).

It is possible to find a pair \( (v_{first}, OPT_{one}) \) in \( \tilde{O}(n + m) \) time with probability at least \( 1 - \frac{1}{2} \).

We will deal with the following problem instead:

**Influence Maximization with Pre-selection (IMP).** Given an \( F \subseteq V \), we want to find a subset \( V' \subseteq V \) with \( |F| = k \) to maximize \( spread(F \cup V') \), where \( k \geq 1 \) is an integer. Furthermore, we also want to report the value of \( spread(F \cup V') \).

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

- \( spread(F \cup V) \geq 0.5 \cdot spread(F^* \cup V) \)
- \( 0.5 \cdot spread(F \cup V) \leq X \leq 1.5 \cdot spread(F \cup V) \).

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 \( 1 - \frac{1}{2} \). 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 $\tilde{O}(\frac{m}{n} \cdot \text{OPT}_{one})$ in expectation and can be obtained in $\tilde{O}(\frac{n}{m} \cdot \text{OPT}_{one})$ expected time after $\tilde{O}(n + m)$ preprocessing time [5, 39]. Repeat the experiment multiple times to produce a collection $R$ of RR-sets.

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 $\{a, b, c, f\}$; if $v = g, R = \{c, g\}$; if $v = a$, 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 $F \subseteq V_0$ 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 $\tilde{O}(|\mathcal{R}| \cdot \text{OPT}_{one})$ in expectation (as mentioned before, each RR-set has an expected size of $\tilde{O}(\frac{n}{m} \cdot \text{OPT}_{one})$).

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

Lemma 6.2. By setting $|\mathcal{R}| = \tilde{O}(kn/\text{OPT}_{one})$, 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($n, m, k$) = $\tilde{O}(n + m + |\mathcal{R}| \cdot \text{OPT}_{one}) = \tilde{O}(n + km)$ 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 $f : 2^{\mathbb{D}} \rightarrow \mathbb{R}_{\geq 0}$ (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 $g : 2^{\mathbb{D}} \rightarrow \mathbb{R}_{\geq 0}$ 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 $\hat{g} : 2^{\mathbb{D}} \rightarrow \mathbb{R}_{\geq 0}$ such that $|g(F) - \hat{g}(F)| \leq \epsilon$ holds for any $F \subseteq \mathbb{D}$ with $|F| \leq k$, where $\epsilon$ is a real value. In this case, we can find a $(1 - 1/e)$-approximate $F$ under $\hat{g}$. namely, $\hat{g}(F) \geq (1 - 1/e) \cdot g(F')$ 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]). $g(F) \geq (1 - 1/e) \cdot \hat{g}(F') - 2\epsilon$.

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

1. Set $\mathbb{D} = \emptyset$;
2. For any $F \subseteq \emptyset$, define $f(F) = \text{spread}(F \cup V_1)$;
3. For any $F \subseteq \emptyset$, define $g(F) = X_{F \cup V_1} \cdot n/|\mathcal{R}|$, where $X_{F \cup V_1}$ is the number of sets $R \in \mathcal{R}$ such that $R \cap (F \cup V_1) \neq \emptyset$;
4. For any $F \subseteq \emptyset$, define $\hat{g}(F) = \hat{g}(F)$.

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

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

(2) simultaneously for all $F \subseteq \emptyset$ 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 $\tilde{O}(\frac{\text{spread}(U)}{\text{spread}(U)})$, with probability at least $1 - 1/n^3$ we have $\text{spread}(U) - (n/|\mathcal{R}|) \cdot X_U \leq 0.05 \cdot \text{spread}(U)$.

The definition of IMP ensures $v_{first} \subseteq V_1$. Hence, for any $F$ in the lemma, $g(F) = \text{spread}(F \cup V_1) \geq \text{spread}\{v_{first}\} = \tilde{O}(\text{OPT}_{one})$. Applying the above fact with $\ell = k + c$ (for some constant $c$ decided later) and $U = F \cup V_1$, we know that $|\mathcal{R}| = \tilde{O}(kn/\text{OPT}_{one})$ 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. □

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 $1 - 1/n^2$.

**Discussion.** A key idea behind our MVA-influence-miseed algorithm is to force $v_{\text{first}}$ into $F$. 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 $O(n^k |V|)$. This is the key behind reducing $|R|$ from $O((k + |V|)/\text{OPT}_{\text{one}})$ to $O(kn / \text{OPT}_{\text{one}})$.

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 $S$ is a $s$-reach (discussed later). We now formulate an instance of the MVA-reachability instance aims to return a solution set $F$ for MVA-reachability. Section 7.2 will prove:

**Lemma 7.1.** If the size of an optimal solution to MVA-reachability is $k^*$, the above algorithm finds an $F$ whose size is larger by a factor of 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 $V_0 \cup \{u\}$; 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.

**Construction.** Let $G$ the weighted graph $G^+$. 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 $| \text{coverage}(u) | \leq n$ 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 $\tau$ 1-vertices. 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, $\text{coverage}(F^* \cup \{s\}) \geq \tau$.

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

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

Assume, on the contrary, that $|\text{MRF}(u)| > |F^*|$. 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 $G_{V_0 \cup F^*}$. 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 $\text{cost}(S) = O(\log \tau) \cdot \text{cost}(F^*)$. Regarding $F$, we have $|F| = \sum_{u \in S} |\text{MRF}(u)| \leq \sum_{u \in S} |\text{MRF}(u)| = \text{cost}(S)$. We thus conclude that $|F| \leq \text{cost}(S) = O(\log \tau) \cdot \text{cost}(F^*) \leq O(\log \tau) \cdot |F^*|^2$, 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 $\pi$ the monotone condition that defines the underlying MVA problem. Recall that a solution refers to a set $F \subseteq V_0$ such that $\pi(G_{V_0 \cup F}) = \text{true}$.
We use a solution tree \( T \) to encode all possible solutions. \( T \) is a perfect binary tree with \( 2|V_0| \) leaves. Every edge of \( T \) is labeled with (i) a vertex in \( V_0 \) and (ii) a status: \text{alive} or \text{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 \( T \), denote by \( A_z \) (or \( D_z \)) the set of alive (or dead, respectively) vertices on the path from the root of \( T \) 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 a solution tree on \( V_0 = \{ a, b, c \} \). Node \( z_5 \) has branching vertex \( c \), \( A_{z_5} = \{ b \} \), and \( D_{z_5} = \{ a \} \).

The next function. We can characterize \( T \) 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 \( T \) is constructed, imagine growing \( T \) in a top-down manner: after creating an internal node \( z \), we set \( v = \text{next}(A_z, D_z) \) 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\((A, D)\)
1. if \( A \cup D = V_0 \) then return NULL
2. \( v = \text{next}(A, D) \)
3. \( T \leftarrow \text{build-solution-tree}\((A \cup \{v\}, D)\) \)
4. \( T \leftarrow \text{build-solution-tree}\((A, D \cup \{v\})\) \)
5. create a node \( z \) with \( T \) (or \( T \)) as the left (or right) subtree and \( v \) as the branching vertex
6. return the tree rooted at \( z \)
```

To construct \( T \), 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\((\emptyset, \emptyset) = b \). Likewise, it can be inferred from node \( z_1 \) that next\((\{b\}, \emptyset) = a \), and from \( z_2 \) that next\((\emptyset, \{b\}) = c \).

The next\((A, D)\) function lends itself nicely to a greedy approach. 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: under such circumstances, which should be the next vertex to flip? The answer is precisely the output of next\((A, D)\).

Traversing the solution tree. Fix a function next\((A, D)\) (i.e., a greedy strategy) and thus also a solution tree \( T \). We perform a pre-order traversal of \( T \). 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 \( \pi(G_{A_z \cup V_1}) = \text{true} \), the subtree of \( z \) can be pruned, because \( A_z \) cannot have fewer 0-vertices than \( A_0 \) for any descendant \( z' \) of \( z \).
- **(Fact 2)** Let \( F_{\text{new}} \) be our best solution so far. If \( A_z \) contains at least \( |F_{\text{new}}| \) 0-vertices, the subtree of \( z \) can be pruned.
- **(Fact 3)** Consider \( z \) as the right child of its parent. The subtree of \( z \) can be pruned if \( \pi(G_{V_1 \setminus D_z \cup V_1}) = \text{false} \). 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. The main observation is that a node \( z \) in \( T \) 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 \( T \) if \( |D_z| > \lambda \), where \( \lambda \geq 0 \) is a constant.

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

**Lemma 8.1.** At most \( (k_{\text{first}} + 1)^{k+1} \) nodes in \( T \) 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 \( \tau \) 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 \( s \leadsto v \leadsto t \) such that (i) the path has length at most \( \tau \), (ii) the intermediate vertices on \( s \leadsto v \) are all in \( A \cup V_1 \), and (iii) the path \( v \leadsto t \) has few 0-vertices.

For this purpose, define \( s_{\text{path}}(v, t) \) as an arbitrary shortest path in \( G \), among all the paths from \( v \) to \( t \) that do not pass any vertex in \( D \). Let \( s_{\text{path}}(v, t) \) be the length of \( s_{\text{path}}(v, t) \), and \( s_{\text{len}}(v, t) \) the number of 0-vertices on \( s_{\text{path}}(v, t) \). Call a 0-vertex \( v \notin (A \cup D) \) a candidate
if \( v \) has an in-neighbor \( u \in (A \cup V_1) \) satisfying \( spdist_{A \cup V_1}(s, u) + \omega(u, v) + spdist_{A \cup V_1}(v, t) \leq \tau \) 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 \( zcnt_{\pi}(v, t) \).

**MVA-connectivity.** Let \( s_1, s_2, \ldots, s_k \) 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 \( x \geq 1 \) be the smallest integer such that \( G_{A \cup V_1} \) includes \( s_1, \ldots, s_x \) in the same CC. Call \( \nu \in V_0 \) a candidate if \( \nu \) (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 \( \nu \) with the smallest \( zcnt_{\pi}(\nu, s_{x+1}) \), where \( zcnt_{\pi}(\nu, s_{x+1}) \) has the same meaning as in MVA-shortest-distance.

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

While \( |R_{next}| \) is set to a constant (10000 in our experiments), we adopt an incremental approach for the size of \( R_\pi \). To ensure success probability at least \( 1 - 1/n \), (as a corollary of Lemma 6.4) we only need to set \( |R_\pi| = \tilde{O}(\log n \cdot n/\text{OPT}_{one}) \) where \( n \) is the number of calls to the \( \pi \) function, and \( \text{OPT}_{one} \) is obtained together with \( \nu_{first} \). This way, \( |R_\pi| \) only needs to increase slowly with time. We append new RR-sets to \( R_\pi \) 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 \( \tau \) vertices in \( G_{A \cup V_1} \). To implement \( next(A, D) \), we define a candidate as a 0-vertex \( \nu \) satisfying (i) \( \nu \notin A \cup D \), (ii) \( \nu \) is adjacent in \( G \) to at least one vertex \( t \) that \( s \) can reach in \( G_{V_1 \cup \{A\}} \) with a 1-path. The function \( next(A, D) \) returns the candidate \( \nu \) with the largest \( \text{coverage}(\nu) \setminus \text{coverage}(A \cup \{s\}) \) (see Section 7.1 for the definition of coverage).

## 9 EXPERIMENTS

**Data graphs.** Our evaluation was based on four graphs: (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) US Patent (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 by first calculating the page rank \( \rho(v) \) of each vertex \( v \). Then 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 US Patent, 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 of influence maximization \([8, 13, 21, 38, 41]\), we set the influence probability \( p_e \) of an edge \( e = (u, v) \) to \( 1/d^{\ast}(v) \) where \( d^{\ast}(v) \) is the in-degree of \( v \).

No special preprocessing was necessary for MVA-reachability.

**Competing methods.** Sections 4-7 have presented an algorithm for each MVA problem. We call those algorithms collectively as Specialized. 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 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 \( s \) to \( t \) in ascending order of length. The algorithm finishes at discovering a path with length greater than \( \tau \), and solves the problem 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 approximation ratio 1.39, and returns the steiner tree as a solution. For MVA-influence-minseed, Baseline is the influence maximization algorithm in \([15]\). It doubles the parameter \( k \) of influence maximization until the subset \( U \) found has \( \text{spread}(U) \geq \tau \); this \( U \) is thus returned as a solution. For MVA-reachability, Baseline first obtains the set \( S \subseteq V \) of vertices reachable from \( s \) (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 \( u \) (assigning length 1 to every edge).

**Environments.** The experimentation was performed on a machine equipped with an Intel 2.3 GHz Dual-Core CPU and 8 GB memory.

**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. 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 \( \tau \) at 1000/m. Note that the
value of $\tau$ is commensurate with the edge weights, which (by the generation 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 $|V|$ $\tau$. 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 smaller $|V|$ results in 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 $|V|$ because its next function computes the shortest-path distances from $s$ to all vertices in $V$. 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 $|V|$ to 0.5n while increasing $\tau$. 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. On the one hand, the value of $k^*$ goes down, which helps reduce time (see Lemma 4.2). On the other hand, a greater $\tau$ 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 $\tau$ (which leads to smaller $k^*$) because its cost is proportional to $k^*$. Figures 5e-g omitted Baseline’s running time (over 500 seconds).

We then turned to MVA-connectivity. In each experiment, $S$ equalled any of the size-$|S|$ subsets of $V$ with the same probability. In Figures 5a-d, we set $|S| = 50$ and measured the performance of Specialized, Uni-first, and Baseline as a function of $|V|$; in Figures 5e-h, we fixed $|V|$ = 0.5n and measured their performance as a function of $|S|$. The solutions from Uni-first were nearly as small as those of Specialized. 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 OPTone (roughly the spread of the most influential vertex in $G$). We varied $\tau$ from 2OPTone to 6OPTone for two considerations. First, this essentially requires the returned solution to have a spread several times that of the most influential vertex. Second, such $\tau$ led to solutions with dozens of vertices, a common practice in the influence-maximization literature (see, e.g., [1, 8, 9, 13, 38, 41]). We set the parameter $|R|$ of Uni-first to 10000. The parameter controls the quality and efficiency of the next function. Ideally, its value should vary according to the graph size, but this requires careful
They significantly outperformed \textit{Uni-first} in all cases. Regarding efficiency, \textit{Uni-first} outperformed \textit{Specialized} on three graphs, while \textit{Baseline} was consistently slower than \textit{Specialized} in all cases.

Let us now attend to MVA-reachability. Figures 8a-d compare \textit{Specialized}, \textit{Uni-first}, \textit{Baseline} under different $|V_1|$ but the same $\tau = 40\text{OPT}_{max}$: Figures 7e-h shows another comparison under different $\tau$ but the same $|V_1| = 25$. On larger graphs, \textit{Specialized} was noticeably superior to its competitors in solution size. Regarding efficiency, \textit{Uni-first} outperformed \textit{Specialized} on three graphs, while \textit{Baseline} was consistently slower than \textit{Specialized} in all cases.

Self-improving behavior of \textit{Universal}. The second set of experiments aims to demonstrate what happens when \textit{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 \textit{Uni-first} as a reference. To explain, suppose that \textit{Universal} finds the first solution in $C_{\text{first}}$ 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 $(1 + \frac{1}{i}) \cdot C_{\text{first}}$ seconds. Doing so till termination will produce a non-ascending sequence: $|F_1|, |F_2|, |F_3|, \ldots$. Given an input and some fixed parameters, we launch $20$ independent runs and average their sequences into an \textit{aggregated} sequence $\Sigma$ as follows. First, set $I_{\text{max}}$ to the length of the 5th longest sequence (of the 20 runs). Then, generate an $\Sigma$ of length $I_{\text{max}}$ where $\Sigma[i] (1 \leq i \leq I_{\text{max}})$ is the mean $|F_i|$ of all the runs. The $\Sigma$ thus obtained manifests the self-improving ability of \textit{Universal}. Also, denote by $C_{\text{avg}}$ the 20 runs’ average $C_{\text{first}}$ 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 $\lambda$ controls the polynomial core (Lemma 8.1). In each diagram, the $x$-axis is the elapsed time, with the first value equal to $C_{\text{first}}$; the $y$-axis is the solution size. For each aggregated sequence $\Sigma$, its $\Sigma[i] (i \geq 1)$ is the $y$-value of the corresponding curve at $x = (1 + \frac{1}{i}) \cdot C_{\text{first}}$. Note that the three curves of each diagram start from the same $y$-value, which is the size of the output of \textit{Uni-first}. The average solution size of \textit{Specialized} is indicated under each figure. In the same fashion, Figures 9e-h present the results for MVA-connectivity with $|V_1| =$
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 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 r [44], (iii) contains at least r occurrences of a user-defined pattern graph [36], (iv) can be converted to a pattern graph using matrix operations [6, 11], etc.

ACKNOWLEDGMENTS

This work was partially supported by GRF grant 14207820 from HKRGC and a research grant from Alibaba Group.

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) ≤ minu∈N(v) spdist(u,k) + w(u, v) is trivial. Next, we will show spdist(v,k) ≥ minu∈N(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 $dist(s)k = 0$, 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 $spdist(v)k \leq dist(v)k$. 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 $spdist(u,k_u)$.

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.

The claim is true if $\overline{u}$ must be succeeded by a vertex $\overline{v}$ in $P$ that is different from $v$. By the inductive assumption, $dist(\overline{v},k_{\overline{v}}) = spdist(\overline{v},k_{\overline{v}})$ when removed. Our algorithm then invoked $\text{relax-edge}$ on $\overline{v}$, leaving $\overline{w}(\overline{v},\overline{u}) = spdist(\overline{v},k_{\overline{v}})$.

This implies $dist(\overline{u},k_{\overline{u}}) \leq dist(v,k)$, 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 $\text{relax-edge}$ to be called on $u$, $v$, $k_u$, leaving $dist(v,k) \leq spdist(u,k_u) + w(u,v) = spdist(v,k)$. 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 $\leq$.

**Proof.** Assume, for contradiction, that $dist(v,k)$ is removed before $dist(v',k')$, but $dist(v',k') \leq dist(v,k)$. Clearly, $v \neq v'$. 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 $dist(s)k$ is the first removed and $v \neq v'$.

Let $\overline{v}$ be the successor of $\overline{u}$ on $P'$. After removing $dist(\overline{v},k_{\overline{v}})$, our algorithm invoked $\text{relax-edge}$ on $\overline{v}$, leaving $\overline{w}(\overline{v},\overline{u}) = spdist(\overline{v},k_{\overline{v}})$.

By the inductive assumption, $\overline{w}(\overline{v},\overline{u}) = spdist(\overline{v},k_{\overline{v}})$.

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

**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 $dist(\overline{u},k_{\overline{u}})$ has 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 $dist(\overline{v},k_{\overline{v}})$, the algorithm invoked $\text{relax-edge}$ on $\overline{v}$, leaving

$$dist(\overline{v},k_{\overline{v}}) \leq dist(t,k^*) \leq \tau,$$

where the first equality is due to Lemma B.1. The algorithm terminated while having $dist(\overline{v},k_{\overline{v}}) \leq \tau$. 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 $dist(t,k^*)$ under $\leq$, contradicting the algorithm’s choice to remove $dist(t,k^*)$ from $Q$.
- If $k' > k^*$, then by $k_{\overline{u}} \leq k^* < k'$ we know that $dist(\overline{u},k_{\overline{u}}) \leq dist(t,k')$, again contradicting the fact that the algorithm chose to remove $dist(t,k')$ from $Q$.

B 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 $\leq$, and that every such value can be removed at most once. Hence, the total number of calls to $\text{relax-edge}$ is at most $k^*$.

With $Q$ implemented as a binary search tree, every call takes $O(1)$ time. The total cost of all those calls is bounded by $O(k^*m)$.

**C PROOF OF LEMMA 6.1**

We will prove the lemma by assuming that $\psi_{\text{final}}$ is a vertex in an optimal solution to the MVA-influence-minused problem. If not, it is straightforward to adapt the subsequent argument by accounting for this single vertex.

We tackle MVA-influence-minused as follows:

1. $k \leftarrow 1$
2. while $k \leq n$ do
3. $(U_k, X_k) \leftarrow$ a 0.5-approx IMP solution under para. $k$
4. if $X_k \geq \tau/4$ 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 $1 - \frac{\log n}{\tau^2} = 1 - O(1/n^2)$.

Let $k^*$ be the value of $k$ at termination. It is clear that

$$\text{spread}(U_k \cup V) \geq \frac{\overline{X}_k}{1.5} \geq \frac{\tau/4}{1.5} = \Omega(\tau).$$
We will reduce the NP-HARDNESS OF MVA-REACHABILITY instance for \( T \) to argue that if all can reach \( G \), then \( T \) has the smallest size among all the covering sub-collections of sets \( S \) for each element in \( S \), and the set vertex \( \tau \). Define \( T^* \) as the smallest \( \tau \) of vertices (including itself) after flipping the vertices in \( F_T \). This means that at least \( \tau^* - |U| = |T^*| + 1 \) of those 1-vertices must be the set vertices in \( F_T \). Therefore, \( |F_T| \geq |T^*| + 1 \), giving a contradiction.

### PROOF OF LEMMA 8.1

We consider an alternative question. Let \( \mathcal{F} \) be a perfect binary tree. Mark a node \( z \) in \( \mathcal{F} \) if the root-to-\( z \) path in \( \mathcal{F} \) makes at most \( k_{\text{first}} \) left turns (Facts 1 and 2) and at most \( \lambda \) right turns (bounded-death heuristic). The question is: what is the number of marked nodes in \( \mathcal{F} \)?

Next, we will show that the answer is no more than \( (k_{\text{first}} + 1)^{\lambda+1} \), which will establish the lemma.

A simple inductive argument suffices. The claim is obviously correct for \( \lambda = 0 \). Assuming correctness for \( \lambda = c + 1 \), next we prove the claim for \( \lambda = c \). Let \( z_{\text{root}} \) be the lowest marked node on the leftmost path of \( \mathcal{F} \). Denote by \( P \) the path from the root to \( z_{\text{root}} \). 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 \((1 + k_{\text{first}})^{\lambda+1}\). The claim thus follows from the fact that \( P \) has \( k_{\text{first}} + 1 \) nodes.

### REFERENCES


