# Network Resilience and the Length-Bounded Multicut Problem: Reaching the Dynamic Billion-Scale with Guarantees

ALAN KUHNLE, VICTORIA G. CRAWFORD, and MY T. THAI, University of Florida, USA

Motivated by networked systems in which the functionality of the network depends on vertices in the network being within a bounded distance $T$ of each other, we study the length-bounded multicut problem: given a set of pairs, find a minimum-size set of edges whose removal ensures the distance between each pair exceeds $T$. We introduce the first algorithms for this problem capable of scaling to massive networks with billions of edges and nodes: three highly scalable algorithms with worst-case performance ratios. Furthermore, one of our algorithms is fully dynamic, capable of updating its solution upon incremental vertex / edge additions or removals from the network while maintaining its performance ratio. Finally, we show that unless $NP \subseteq BPP$, there is no polynomial-time, approximation algorithm with performance ratio better than $\Omega(T)$, which matches the ratio of our dynamic algorithm up to a constant factor.

Additional Key Words and Phrases: Scalable algorithms; length-bounded multicut

**ACM Reference Format:**
Alan Kuhnle, Victoria G. Crawford, and My T. Thai. 2018. Network Resilience and the Length-Bounded Multicut Problem: Reaching the Dynamic Billion-Scale with Guarantees. *Proc. ACM Meas. Anal. Comput. Syst.* 2, 1, Article 4 (March 2018), 26 pages. https://doi.org/10.1145/3179407



## 1 INTRODUCTION

Connectivity in a graph has historically been an important metric of the functionality of a network; that is, the service provided by the network is extant between two nodes if there exists a path between these two nodes. This consideration has led to the study of many forms of cutting problems in a network: *e.g.* the minimum cut problem, the minimum multicut problem and the sparsest cut problem, among many others [1]. In addition, various measures of connectivity have formed the basis for assessment of a network's vulnerability to external perturbation [2, 3].

Transcending connectivity, many network applications depend on some measure of network distance between a pair of connected nodes. An edge weight representing a metric related to network functionality may be associated to each edge, and a pair of nodes only benefits from the network if the weighted, shortest-path distance between the pair is below a threshold $T$. For example, in an Industrial Internet-of-Things (IIoT) network [4], Quality-of-Service (QoS) metrics (*e.g.* packet loss, time delay) between a pair in the network are critical to many applications, and communication protocols have been developed to guarantee a threshold of QoS between communicating pairs [5]. As another example, consider a time-sensitive delivery on a road network, weighted by the travel time between destinations. Mere connectivity between a source and destination is insufficient when

Authors' address: Alan Kuhnle, kuhnle@ufl.edu; Victoria G. Crawford, vcrawford01@ufl.edu; My T. Thai, mythai@ufl.edu, University of Florida, Computer & Information Science & Engineering, Gainesville, FL, 32306, USA.


2476-1249/2018/3-ART4 $15.00
https://doi.org/10.1145/3179407

a guarantee on the delivery time is desired; such guarantees are offered by popular retailers such as Amazon [6].

Consequently, a natural question is which edges or links in the network are critical for the network functionality between a desired set of pairs [7]. With this motivation, we consider the length-bounded multicut problem (LB MULTICUT), in which a weighted[1], directed graph $G$, threshold $T$, and set of pairs $\mathcal{S}$ is given, and the problem is to identify a minimum-size set of edges whose removal ensures the weighted, shortest-path distance between each pair in $\mathcal{S}$ is greater than the threshold $T$. Intuitively, the goal of this problem is to assess how robust the network is; the larger the size of the set of edges found, the more resilient the network is to perturbation in terms of edge failure; in addition, membership in this set of edges provides an indication of the importance of an edge to the desired functionality.

In the context of network reliability, the node version of the LB MULTICUT problem was studied by Kuhnle *et al.* [7], who formulated several algorithms that are fixed-parameter tractable (FPT) with respect to the parameter $T$; that is, they run in polynomial time if $T$ is considered to be a constant. However, these algorithms do not scale beyond tens of thousands of edges and nodes, as we demonstrate in Section 4. Modern networked systems are increasingly massive in scale [8], often with sizes of at least millions of vertices and edges. For example, one network of autonomous systems in the internet obtained from traceroutes has more then 12 million vertices and edges [9]. A simple, primal-dual algorithm[2] for LB MULTICUT does scale to large networks, but its solution quality in practice is often far from optimal, as we demonstrate in our experimental evaluation. Furthermore, modern networks rapidly change as nodes and links enter and leave the network [4, 9–11]. Hence there is a need for dynamic algorithms that can efficiently update their solution in response to changes in the network, rather than recomputing from scratch on the modified network.

*Contributions.* Given an instance $(G, d, \mathcal{S}, T)$ of LB MULTICUT, let $n$ be the number of vertices in $G$, $m$ be the number of edges, and $T_0 = T/d_0$, in which $d_0$ is the minimum edge weight on $G$.

- We provide three highly scalable algorithms for LB MULTICUT: (1) SAP, a probabilistic, FPT-approximation algorithm with performance ratio $O(T_0 \log n)$ with probability at least $(1-1/m)$; (2) MIA, a multicut-based algorithm with performance ratio $O(Mn^{11/23})$, in which $M$ is the number of directed multicuts employed by MIA; and (3) TAG, an approximation algorithm with performance ratio $T_0$. Here, the *performance ratio* is the maximum ratio over all problem instances of the size of the set of edges returned by the algorithm divided by the optimal size.
- In the presence of dynamic edge / vertex insertions and deletions to the network, the dynamic version of TAG can efficiently update its solution in response to these incremental changes, while maintaining its performance guarantee of $T_0$. Thus, our algorithm TAG is fully dynamic, which leads to massive speedups over static TAG.
- When $T$ is fixed and edge weights are uniform, the previous best lower bound on the approximability of LB MULTICUT is $\Omega\left(\sqrt{T}\right)$ [12] assuming the Unique Games Conjecture. We improve this lower bound to $\Omega(T)$ unless $NP \subseteq BPP$ [3].
- We extensively evaluate our algorithms on large-scale, real-world networks. All three of our algorithms are demonstrated to scale to networks with billions of edges in under a few hours and to return nearly optimal solutions, up to a factor of 1000 and usually a factor of at least 5

[1]Each edge $e \in G$ has a positive weight $d(e)$

[2]We present and analyze the primal-dual approach to LB MULTICUT in Section 2.

[3]Bounded-error probabilistic polynomial time: class of decision problems solvable by a probabilistic Turing machine in polynomial time with an error probability bounded away from 1/2 [13].

smaller than the primal-dual algorithm. The source code of our implementation is available [14].

*Related work. Length-bounded cuts.* For the case that the target set $\mathcal{S}$ contains a single pair and edge weights are uniform, length-bounded cuts and flows were studied extensively by Baier *et al.* [15]. In their work, the problem is proven to be NP-hard and a $T/2$-approximation algorithm is given. This algorithm cannot be directly applied to the LB MULTICUT problem; our algorithm MIA generalizes it through the novel concept of compatible paths (Section 3.2). The hardness results for the length-bounded, edge cut were improved in Lee [12], wherein an $\Omega\left(\sqrt{T}\right)$-hardness result is shown under the Unique Games Conjecture (UGC). In this work, we show that when multiple pairs are allowed, a stronger lower bound of $\Omega(T)$ can be proven under the much weaker assumption $NP \nsubseteq BPP$.

The parameterized complexity of the length-bounded cut problem was studied in Golovach *et al.* [16], wherein the parameterization is with respect to the size $c$ of the cut and the length $T$ of the paths. They provide an FPT-exact algorithm with running time $O(T^{c+1}m)$ and analyze the relationship among related problems in terms of FPT reducibility. The parameterized complexity is further studied in Dvorak *et al.* [17], who also consider the complexity of the length-bounded multicut problem in addition to the case of a single pair. Neither of these works consider approximation algorithms, which are the focus of this paper.

Kuhnle *et al.* [7] have studied the node version of the LB MULTICUT problem. They provide two FPT-approximation algorithms [18], GEN and FEN, wherein the parameterization is with respect to the length $T$ of the paths. These algorithms require a listing of all paths of lengths at most $T$ and hence do not scale to large networks, as shown in our experimental evaluation (Section 4) wherein we compare to our adaptation of GEN to the edge version of LB MULTICUT. They also provide a greedy, sampling-based approximation GEST which has a probabilistic, bicriteria ratio of $O(\alpha\Delta^{T_0} + \log k)$, where $\alpha$ is a parameter in $(0, 1)$, $\Delta$ is the maximum degree in the graph, and $k$ is the number of pairs in the problem instance. Since in the worst case $\Delta = n - 1$, this performance guarantee can be almost as bad as the trivial ratio $n$. Although it is offset by $\alpha$, the running time of GEST is proportional to $k^3/\alpha^2$; therefore, to meaningfully ameliorate the $\Delta^{T_0}$ factor by choosing a small $\alpha$ would make the running time prohibitive. Additionally, in practice, Kuhnle *et al.* had to augment this approach with a heuristic due to difficulty obtaining valid path samples.

Our algorithm SAP is inspired by GEST in that we use a greedy approach based upon path sampling; however, we boost the number of valid path samples by using probabilistic hints based upon shortest-path computations to guide the sampling. Because of these biased path samples, we found it unnecessary to use any heuristic to obtain valid samples in practice. Furthermore, we prove a stronger performance guarantee of $O(T_0 \log n)$ for SAP, which holds with probability $1 - 1/m$. Finally, our implementation of SAP is scalable to networks with billions of edges, as shown in our experimental evaluation.

*The shortest-path network interdiction problem.* In the case of a single pair of vertices $(s, t)$, the *shortest-path network interdiction* (SPNI) problem is defined as follows: given a budget of size $B$, remove $B$ edges from a graph to maximize the distance $d(s, t)$. While related to the single-pair LB MULTICUT problem (*i.e.* LB CUT), the objective is to maximize the distance $d(s, t)$ rather than to minimize the size required before the distance is above a threshold. In terms of approximability, we are unaware of any approximation algorithms for SPNI.

An exact algorithm for SPNI was formulated by Malik *et al.* [19]. The problem was formulated as a bilevel mixed-integer program by Israeli *et al.* [20], who explored exact solutions to the problem using Benders decomposition. Generalizations of the problem and this exact algorithm to more general Attacker-Defender games have been studied by Brown *et al.* [21] and Yates *et al.* [22].

However, these exact approaches are suitable only for tiny networks; furthermore, any multi-pair generalization of SPNI has not been studied, to the best of our knowledge.

*Classical multicut problem.* The LB MULTICUT problem generalizes the classical multicut problem, wherein given a set of $k$ pairs $\mathcal{S}$, the problem asks for the minimum number of edges (or nodes) whose removal ensures each pair in $\mathcal{S}$ is topologically disconnected. Results on this problem depend on whether the node version or edge version is considered and whether the graph is undirected or directed. For the edge version in undirected graphs, an $O(\log k)$-approximation was developed by Garg *et al.* [23] by considering multicommodity flow. In directed graphs, Gupta [24] developed an $O(\sqrt{n})$-approximation algorithm, which was later improved to $O(n^{11/23})$ by Agarwal *et al.* [25]. Our algorithm MIA employs an approximation algorithm for directed edge multicut on subproblems constructed to ensure that such a multicut lower bounds the optimal solution of LB MULTICUT; the performance ratio of MIA is the number $M$ of such multicuts it requires times the approximation ratio of the directed multicut algorithm employed. Despite the fact that these multicut algorithms employ an LP solution via the ellipsoid method and are not scalable to large networks, the multicut instances required by MIA are small enough that we found MIA to be surprisingly scalable in practice.

*Technical contributions.*

- By using a biased self-avoiding random walk in Section 3.1, we were able to use fewer samples in practice to estimate the number of paths upon which an edge lies than previous approaches [7, 26]. The improved practical performance of this estimator may be independently useful for other applications that require this value.
- For the algorithm MIA (Section 3.2), we introduce a notion of *compatible paths*, which is a set of paths such that no path longer than any in the set can be created from the union of subpaths of paths in this set. This notion could be of independent interest for problems involving bounded-length paths.
- For the algorithm TAG (Section 3.3), we use a primal-dual solution to bound the worst-case performance of TAG under incremental graph changes and improve this solution in practice by periodic pruning. This dynamic approach could be useful for other combinatorial problems on dynamic networks. As an example, consider subgraph transversal problems, where a minimum-size set of edges having non-empty intersection with all subgraphs of a specified type is desired; LB MULTICUT is a problem of this type.

*Organization.* The rest of this paper is organized as follows. In Section 2 we formally define the problem and discuss the primal-dual approach, and our inapproximability result is presented in Secton 2.2. In Sections 3.1, 3.2, and 3.3, we present and analyze SAP, MIA, and TAG, respectively. In Section 4, we evaluate our algorithms and compare to previously developed algorithms for LB MULTICUT.

## 2 DEFINITIONS AND INAPPROXIMABILITY

In this section, we formally define the length-bounded multicut (LB MULTICUT) problem considered in this work. We also present an IP formulation of the problem and present the primal-dual algorithm to approximate the LB MULTICUT problem. We prove our inapproximability result in Section 2.2.

Problem 1 (Min. length-bounded multicut (LB MULTICUT)). *Given digraph $G$, positive weight function $d : E \to \mathbb{R}$, threshold $T$, and target set $\mathcal{S} = \{(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)\}$, determine a minimum-size set $S$ of edges such that $d(s_i, t_i) > T$ for all $i$ after the removal of $S$ from $G$, in which $d(s, t)$ is the $d$-weighted shortest paths distance from $s$ to $t$. A problem instance may be represented by the tuple $(G, d, \mathcal{S}, T)$.*

*IP formulation.* A path $p = p_0 p_1 \ldots p_l \in G$ is a sequence of vertices such that $(p_{i-1}, p_i) \in E$ for $i = 1, \ldots, l$. A *simple path* is a path containing no cycles (*i.e.* repeated vertices); the length of a path

Table 1. Notation

| Notation | Definition |
|---|---|
| $G = (V, E, d)$ | Simple, directed graph, with edge weights $d$ |
| $V(G), E(G)$ | Vertex and edge sets of digraph $G$, respectively |
| $n(G), m(G)$ | Number of vertices, edges in $G$, respectively |
| $k$ | The number of pairs in target set $\mathcal{S}$ |
| $T$ | The threshold or bound on the path length |
| $d_0$ | Minimum edge weight in problem instance |
| $T_0$ | $T/d_0$ |
| $E(p)$ | Set of edges on the simple path $p$ |
| $d(p)$ | The sum of the edge weights of edges in path $p$ |
| $\Delta$ | $\max\{\text{out-degree}(s) + \text{in-degree}(s) : s \in G\}$ |
| $\Gamma$ | $\max\{\text{out-degree}(s), \text{in-degree}(t) : (s,t) \in \mathcal{S}\}$ |
| $\mathcal{P}, \mathcal{P}(G,d,T)$ | Set of $T$-bounded paths in $G$ between pairs in $\mathcal{S}$ |
| $f : X \to Y$ | $f$ is a function from set $X$ to set $Y$ |
| $G \backslash S$ | The digraph $(V, E \backslash S)$ |
| $\mathcal{U}$ | Set of edge-disjoint paths maintained by TAG |
| $OPT$(instance) | Optimal solution to problem instance |
| $\gamma \in (0,1)$ | Bias parameter in the sampling of SAP |

is the sum of its edge weights. Let $\mathcal{P}_{s_i}^{t_i}(G,d,T)$ denote the set of simple paths $p$ in digraph $G$ from $s_i$ to $t_i$ that satisfy the condition $d(p) \le T$, in which $d(p) = \sum_{i=1}^{l} d(p_{i-1}, p_i)$. Any path $p$ for which $d(p) \le T$ is termed *$T$-bounded*. Let $\mathcal{P}(G,d,T) = \bigcup_{i=1}^{k} \mathcal{P}_{s_i}^{t_i}(G,d,T)$; when $G$,$T$, and $d$ are clear from context, we write $\mathcal{P} = \mathcal{P}(G,d,T)$.

The problem LB MULTICUT can be formulated as the following integer program (IP 1), where $x(e)$ represents whether edge $e$ is chosen into the solution set $S$:

$$\min \sum_{e \in E} x(e), \qquad \text{such that}$$

$$\sum_{e \in p} x(e) \ge 1, \qquad \forall p \in \mathcal{P} \tag{1}$$

$$x(e) \in \{0,1\}, \qquad \forall e \in E \tag{2}$$

In the following, we will refer to LP 1, the linear relaxation of IP 1, in which constraints (2) are replaced with $0 \le x(e)$.

*Discussion.* If $T$ and the minimum edge weight $d_0$ are regarded as fixed parameters, it is possible to list all paths in $\mathcal{P}$ in polynomial time and hence also solve LP 1 in polynomial time. However, this path listing is very expensive in practice, since it requires $\Omega(n^{T_0})$ time in the worst case, in which $T_0 = T/d_0$. Even in the case in which $T$ is a fixed parameter and edge weights are uniform, LB MULTICUT is NP-hard as shown in Baier *et al.* [15] for the case when $\mathcal{S}$ consists of a single pair.

Our algorithms SAP, TAG, and MIA are designed to be efficient even when $T$ is large and hence do not require a listing of $\mathcal{P}$ or a solution to LP 1.

*The primal-dual algorithm.*[4] The LB MULTICUT problem can be efficiently tackled by the following primal-dual algorithm. Pick any pair $(s,t) \in \mathcal{S}$, such that $d(s,t) \le T$, and compute a shortest-path $p$

[4]For a detailed treatment of the primal-dual approach to the design of approximation algorithms, we refer the reader to the textbook [1] by Vazirani.

between $s, t$. Add all edges of this path $p$ to the solution $S$. Repeat this process until all pairs satisfy $d(s,t) > T$.

After the primal-dual algorithm, the edges of $S$ obtained will form a union of edge-disjoint paths, each with at most $T_0$ edges; denote this set of edge-disjoint paths as $\mathcal{U}$. Since any optimal solution must choose at least one edge in each path $p \in \mathcal{U}$ and these paths are edge-disjoint, the primal-dual algorithm has a performance ratio of $T_0$ to the optimal size. By our inapproximability result in Section 2.2 there is no better worst-case guarantee up to a constant factor unless $NP \subseteq BPP$.

Although the primal-dual algorithm is efficient and able to run on large networks, it often performs far from the optimal solution as we show in our evaluation in Section 4. To upper bound its solution, our algorithm TAG employs a set of edge-disjoint paths $\mathcal{U}$ corresponding to a primal-dual solution, which allows TAG to maintain the same worst-case guarantee of $T_0$ but perform close to the optimal size in practice. Finally, we show how to efficiently update the solution of TAG in response to changes in the network while maintaining the same performance guarantee.

*Node version of the problem.* The node version of the LB MULTICUT problem asks for the set $S$ to be a subset of vertices rather than edges in the problem definition above. Both node and link resiliency are of interest; Kuhnle *et al.* [7] studied primarily the length-bounded node cutting problems. Our algorithms TAG and SAP can be easily adapted for the node version, as we adapt the algorithm GEN of [7] to the edge version of the problem for comparison in Section 4. Our algorithm MIA changes more significantly when adapted to the node version, since there is not yet an approximation algorithm for directed node multicut.

### 2.1 Motivation and applications

In this section, we give brief overviews of two potential applications of LB MULTICUT.

*2.1.1 Industrial Internet of Things.* An emerging application for pseudocut problems is the Industrial Internet of Things (IIoT). As everyday objects become increasingly equipped with means for electronic identification and communication, from Radio Frequency Identification (RFID) to smarter communication capabilities, new applications and scenarios have emerged in the Internet of Things [4, 29].

As surveyed in [30], an emerging trend is to integrate communication capabilities into industrial production systems. Such cyberphysical systems (CPS) in the production process are connected to conventional business IT networks. Integrated CPS allow extensive monitoring and control of production facilities in real time. However, the QoS requirements for control of production systems are very strict, and special routing protocols have been formulated to guarantee acceptable QoS conditions [5]. An IEEE task group on Time-Sensitive Networking (TSN) [31] is currently chartered to provide specifications to allow time-synchronized low latency streaming services through 802 networks. Critical data streams are guaranteed certain end-to-end QoS by resource reservation; this service is intended for industrial applications such as process control, machine control, and vehicles; and for audio/video streams.

As an example application for the LB MULTICUT, consider a set of communicating pairs $\mathcal{S}$ in IIoT as described above. Further, suppose that an acceptable level of packet loss ratio between any pair in $\mathcal{S}$ is $10^{-10}$. Then, the problem instance is the IIoT network $G$, with edges $e$ weighted by the metric $d$ defined in Lemma 2.1 below, the set of pairs $\mathcal{S}$ and $T = 10^{-10}$. A solution $S$ to this instance quantifies the resilience of the network in the sense that the cumulative packet loss ratio of at most $10^{-10}$ can be maintained between at least one pair in $\mathcal{S}$ under any amount of edge failure smaller than $|S|$.

To convert the packet error rate between nodes to an additive metric, we define the following transformation. Given network $G = (V, E)$, let $p_{uv} \in [0, 1]$ represent packet error rate for each

edge $(u, v) \in E$. Then, the transformation is

$$p_{uv} \to -\log(1 - p_{uv}). \tag{3}$$

Lemma 2.1. *Let $p_{uv}$ represent packet error rate between each $(u, v) \in E$. Then the transformation (3) yields an additive metric $d$ such that $1 - \exp(-d(s, t))$ is the lowest cumulative packet error rate between nodes $s, t$ over all possible routing paths.*

Proof. Let $G = (V, E)$ with packet error rate $p_{er}(e) \in (0, 1)$ be given for each $e \in E$. Let $d(e) = -\log(1 - p_{er}(e))$. Let $s, t \in G$, and $\mathscr{P}$ be the set of all paths in $G$ from $s$ to $t$. Then

$$\begin{aligned} d(s, t) &= \min_{p \in \mathscr{P}} \sum_{e \in p} d(e) \\ &= \min_{p \in \mathscr{P}} \sum_{e \in p} -\log(1 - p_{er}(e)) \\ &= -\max_{p \in \mathscr{P}} \log \prod_{e \in p} (1 - p_{er}(e)) \end{aligned}$$

Now, $\prod_{e \in p}(1 - p_{er}(e))$ is the probability a packet is successfully transmitted along path $p$. Thus, maximizing this probability over all paths minimizes both $d(s, t)$ and the cumulative packet error rate between $s, t$.

Furthermore, if packet error rate threshold $P$ is given, then by similar reasoning

$$d(s, t) < -\log(1 - P) \iff p_{er}(s, t) < P,$$

where $p_{er}(s, t)$ is the cumulative packet error rate between $s, t$. □

*2.1.2 Military communications networks.* Next generation millitary communications networks will be multilayer, interdependent networks [32–34] comprising wired fiber-optic and wireless components, including satellite communications. For example, consider the proposed Army Warfighter Information Network-Tactical (WIN-T) network, the theory of operation for which is contained in [33]. WIN-T comprises interdependent wireless and wired components that are organized into layers; the WIN-T multi-tiered architecture is organized as follows: (1) the space layer, utilizing military satellite communications (MILSATCOM) and commercial satellite bands, (2) the airborne layer, consisting of unmanned aerial vehicles (UAVs), (3) the ground layer, which contains many different kinds of nodes. The nodes in these layers communicate to each other in a variety of ways including wired LANs, wireless wide-area networks, and satellite communications.

To ensure QoS in WIN-T, traffic is only admitted to the network when the network infrastructure and congestion state offer a high probability that the traffic can be delivered within QoS requirements specified in WIN-T Baseline Requirements Document. Thus, communication failure between a pair $s, t$ of nodes in the network may occur despite the existence of a routing path between $s$ and $t$ in the network, if any of the QoS metrics are greater than a threshold $T$.

Therefore, the `LB MULTICUT` problem would identify the most critical edges for communication between a given set of node pairs. For example, a commanding node $s$ which communicates with multiple infantry units $\{t_i\}$. If communication between $s$ and $\{t_i\}$ is a high priority, critical edges identified would be especially important to protect against an adversarial attack.

## 2.2 Inapproximability result

In this section, we show that the performance ratio of $T_0$ of `TAG` and the primal-dual algorithm are optimal up to a constant factor unless $NP \subseteq BPP$. The results in this section hold for the special case of `LB MULTICUT` when $T$ is a fixed parameter rather than part of the input; for clarity, we write $T$-`LB MULTICUT` to emphasize that $T$ is a fixed constant in this section.

Theorem 2.2. *Let $T \geq 16$. Unless $NP \subseteq BPP$, there is no polynomial-time algorithm to approximate $T$-LB MULTICUT within a factor of*

$$\left\lfloor \frac{T}{6} \right\rfloor - 1 - \epsilon,$$

*for any $\epsilon > 0$.*

Proof. Fix $r \geq 6$, and let digraph $G$ be an instance of Edge ($\leq r$)-Cycle Transversal ($\leq r$-ECT), the problem to find a minimum set of edges $C$ intersecting every cycle of length at most $r$ in digraph $G$. For this problem, Guruswami *et al.* [35] proved there is no polynomial-time approximation within factor of $\left\lfloor \frac{r}{2} \right\rfloor - 1 - \epsilon$ unless $NP \subseteq BPP$.

Then let $T = 3r - 2$; we reduce instance $G$ to an instance $G'$ of $T$-LB MULTICUT as follows. For each $v \in G$, we add pair of vertices $v_{in}, v_{out}$ to $G'$, and $m = m(G)$ vertices $v_1, \ldots, v_m$ to $G'$; next add edges $(v_{in}, v_i)$, $(v_i, v_{out})$ for all $i = 1, \ldots, m$. Next, for every edge $(u, w)$ in $G$, add edge $(u_{out}, w_{in})$ to $G'$. Finally, create target set $\mathcal{S} = \{(v_{out}, v_{in}) : v \in V(G)\}$. Thus, $G'$ is a valid instance of $T$-LB MULTICUT. Let $C_{opt}$ be an optimal solution on $G$ to $\leq r$-ECT and $C'_{opt}$ be an optimal solution for the instance $G'$ of $T$-LB MULTICUT.

If the removal of edge set $C$ from $G$ breaks every cycle of length at most $r$ in $G$, then removing set $C' = \{(u_{out}, w_{in}) : (u, w) \in C\}$ breaks every path in $\mathcal{P}(G')$. To see this, let $p \in \mathcal{P}(G')$; then $p \in \mathcal{P}^{v_{in}}_{v_{out}}$ for some $v \in V(G)$. Hence $p = v_{out} w^1_{in} w^1_{i_1} w^1_{out} w^2_{in} \ldots w^l_{out} v_{in}$ for some $w^1, \ldots, w^l \in V(G)$, and $d(p) \leq T$. Also, $d(p) = 3l + 1$ implies $l \leq \frac{T-1}{3}$, so $l + 1 \leq r$. Furthermore, by definition of $G'$, $v = w^0, w^1, \ldots, w^l$ is an cycle of length $l + 1$ in $G$, so $(w^i, w^{i+1}) \in C$ for some $i$, and so $(w^i_{out}, w^{i+1}_{in}) \in C'$ and also lies on $p$. As a result, $|C'_{opt}| \leq |C_{opt}|$.

Let $C'$ be a collection of edges in $G'$, the removal of which breaks every path in $\mathcal{P}(G')$. We will construct a set $C = \phi(C')$ from $C'$ that intersects every ($\leq r$)-cycle of $G$, such that $|C| \leq |C'|$. First, notice that by the choice of $\mathcal{S}$, including edge $(v_{in}, v_i)$ is equivalent to including $(v_i, v_{out})$ and there is nothing to be gained by including both edges. Therefore, if an edge in $C'$ is incident with $v_i$ for any $i, v$, let such edge have form $(v_{in}, v_i)$. Second, we argue that if $C'$ contains any edge of form $(v_{in}, v_i)$, it may be removed to create a smaller set of edges that still intersect every path in $\mathcal{P}$. For the first case, suppose $e_j = (v_{in}, v_j) \in C'$ for all $j = 1, \ldots, m$. Then, a smaller set $C'$ can be created by removing set $\{e_j\}$ from $C'$ and replacing with $\{(w_{out}, v_{in}) : w_{out} \in V'\}$. For the second case, suppose there exists $i, j$ such that $e_i \in C'$ and $e_j \notin C'$. For every path $p_i \in \mathcal{P}$ containing $e_i$ there exists an analogous path $p_j$ containing $e_j$, in which $p_i$ and $p_j$ differ only by the nodes $v_i$,$v_j$. Since an edge of $p_j$ not equal to $e_j$ must be contained in $C'$, that edge also lies upon $p_i$ and is contained in $C'$. Hence, $e_i$ may be removed from $C'$ while maintaining $\mathcal{P}(G' \backslash (C' \backslash \{e_i\})) = \emptyset$. Therefore, a feasible solution $C''$ can be constructed from $C'$ in which each edge is of the form $(u_{out}, w_{in})$ for some $u, w \in V(G)$; then, define $\phi(C') = \{(u, w) : (u_{out}, w_{in}) \in C''\}$. By a similar argument as above (except in reverse), $\phi(C')$ must intersect every cycle of length at most $r$ in $G$, and $|\phi(C')| \leq |C'|$. As a result, $|C_{opt}| \leq |C'_{opt}|$.

So we have shown $|C_{opt}| = |C'_{opt}|$. Now, let $\mathcal{A}$ be an approximation algorithm for $T$-LB MULTICUT with ratio $\left\lfloor \frac{T}{6} \right\rfloor - 1 - \epsilon$, for some $\epsilon > 0$. Let $C'_{\mathcal{A}}$ be the result of running $\mathcal{A}$ on $G'$. Then

$$\frac{|\phi(C'_{\mathcal{A}})|}{|C_{opt}|} \leq \frac{|C'_{\mathcal{A}}|}{|C'_{opt}|} \leq \left\lfloor \frac{T}{6} \right\rfloor - 1 - \epsilon \leq \left\lfloor \frac{r}{2} \right\rfloor - 1 - \epsilon.$$

Therefore, the constuction of $G'$ from $G$, algorithm $\mathcal{A}$, and transformation $\phi$ would result in a polynomial time algorithm for $\leq r$-ECT with ratio at most $\left\lfloor \frac{r}{2} \right\rfloor - 1 - \epsilon$, contradicting Guruswami *et al.* [35] unless $NP \subseteq BPP$. Finally, notice that this argument (with the same $r$) works for $T_1 = T + 1$,

$T_2 = T + 2$ as well, since no new paths in $\mathcal{P}(G')$ are created by these increases in $T$. Hence, the result holds for all $T \geq 16$. □

## 3 ALGORITHMS FOR LB MULTICUT

### 3.1 Probabilistic, $O(T_0 \log n)$ FPT Approximation

In this section, we present SAP (Alg. 1), an FPT-approximation algorithm for LB MULTICUT with ratio $O(T_0 \log n)$ with probability at least $1 - 1/m$. SAP runs in polynomial time when the parameter $T_0$ is fixed.

*3.1.1 Overview.* In essence, SAP is a greedy algorithm that attempts, in each iteration, to select an edge $e$ that hits the largest number of paths in $\mathcal{P}$. Rather than an expensive listing of $\mathcal{P}$ to determine $e$, an estimator is employed by a path sampling procedure to select the best edge. This process is repeated iteratively until a feasible solution $S$ is obtained; feasibility is checked by shortest-path computations.

The estimator employed by SAP is adapted from the estimator of GEST in Kuhnle *et al.* [7]; in contrast to GEST, the probability distribution and path sampling procedure are biased towards shorter paths by the parameter $\gamma$, which addresses the issue GEST has in obtaining useful path samples. Also, our performance guarantee for SAP is much stronger than the bicriteria ratio of GEST, as discussed in Section 1 (Related Work).

In Section 3.1.2, we define the estimator $\hat{y}(e)$ employed in each iteration, and in Section 3.1.3, we describe the process for sampling of paths. In Section 3.1.4, we present the algorithm and we prove its performance guarantee in Section 3.1.5.

*3.1.2 Estimator.* Let an instance of LB MULTICUT be given. For each edge $e \in G$, we define $y(e)$ to be the number of paths in $\mathcal{P}$ upon which $e$ lies; *i.e.*, $y(e) = |\{p : e \in p \text{ and } p \in \mathcal{P}\}|$. To estimate $y(e)$, we define estimator $\hat{y}(e)$ in the following way. Let $f$ be any probability distribution on a set of paths $\mathcal{Q}$ containing $\mathcal{P}$, such that for any $p \in \mathcal{P}$, $f(p) > 0$. Let $p_1, \ldots, p_L$ be $L$ paths sampled from $f$. Then we define the estimator $\hat{y}$ as follows:

$$\hat{y}(e) = \frac{1}{L} \sum_{i=1}^{L} \frac{\mathbf{I}\left(e \in p_i \text{ and } p_i \in \mathcal{P}\right)}{f(p_i)}, \tag{4}$$

in which $\mathbf{I}(\cdot)$ is the indicator function returning 1 if its argument is true and 0 otherwise.

Lemma 3.1. *$\hat{y}(e)$ is an unbiased estimator of $y(e)$; that is, $\mathbb{E}[\hat{y}(e)] = y(e)$.*

Proof.

$$\mathbb{E}[\hat{y}(e)] = \mathbb{E}\left[\frac{\mathbf{I}\left(e \in q \text{ and } q \in \mathcal{P}\right)}{f(q)}\right] = \sum_{q \in \mathcal{P}} \mathbf{I}\left(e \in q\right) = y(e). \quad \square$$

*3.1.3 Path-sampling procedure.* We employ biased, self-avoiding random walks in this procedure. First, we select a source-target pair $(s, t)$ uniformly randomly from $\mathcal{S}$. We perform a biased, self-avoiding random walk from $s$ as follows. Let a shortest-path tree directed towards $t$ be given. For each node $v \in V$, let $a(v)$ be the parent of $v$ in this tree. Given a simple path $q$ from $s$ ending at node $c$, we choose the next step with the following probability distribution. Let $N(c) = \{v : (c, v) \in E, v \notin q\}$. If $N(c) = \{v\}$, then $v$ is chosen with probability 1. Otherwise, if $a(c) \in N(c)$, we assign it probability $\gamma$ and probability $(1 - \gamma)/(|N(c)| - 1)$ to the remaining nodes. If $a(c) \notin N(c)$, we assign uniform probability to each node in $N(c)$. If the length of the path exceeds $T$, the path has no valid next steps, or it reaches the target $t$, we stop the walk and return the path.

With the path-sampling procedure defined, we next define the probability distribution $f(q)$ on all paths in $G$. For any path $q$, define the probability $f(q)$ to be the probability that $q$ is sampled by the above procedure. Next, define $\mathcal{Q} = \{q : f(q) > 0\}$; the following lemma guarantees that $\mathcal{P} \subseteq Q$.

Lemma 3.2. *Let $\mathcal{P}(G, d, T)$ be the set of all $T$-bounded paths in $G$ between pairs in $\mathcal{S}$. Then $\mathcal{P} \subseteq \mathcal{Q} = \{q : q$ is a path in $G$ and $f(q) > 0\}$.*

Proof. Let $p \in \mathcal{P}$. Then $p$ is a $T$-bounded path from $s_i$ to $t_i$ for some $(s_i, t_i) \in \mathcal{S}$. There is a nonzero probability that $(s_i, t_i)$ is selected by the sampling procedure. Since $p$ is a simple path with $d(p) \leq T$, the next node of $p$ has a nonzero probability of being selected at each step of the walk, given that any initial segment of $p$ has been selected. Hence, $f(p) > 0$. □

**ALGORITHM 1: S**ampling **AP**proximation (SAP), with ratio $O(T_0 \log n)$ with probability $1 - 1/m$

**Input** : Instance $(G, d, \mathcal{S}, T)$ of LB MULTICUT, $\gamma \in (0, 1)$
**Output**: Solution $S \subseteq E$ to LB MULTICUT

1 $\mathcal{S} = \{(s, t) \in \mathcal{S} : d(s, t) \leq T\}$, $S = \emptyset$;
2 **while** $\mathcal{S} \neq \emptyset$ **do**
3 Let $\Gamma = \max\{\text{out-deg}(s), \text{in-deg}(t) : (s, t) \in \mathcal{S}\}$;
4 Let $L = 2|\mathcal{S}|^2\Gamma^2 \log(2m^3)n^{2T_0}$;
5 Sample $L$ paths $p_1, \ldots, p_L$ according to Section 3.1.3. Compute $\hat{y}(e)$ according to the sampled paths for each $e \in E$ by Eq. 4;
6 Choose $e' = \arg\max \hat{y}(e)$ into $S$;
7 $G = G \backslash \{e'\}$, $\mathcal{S} = \{(s, t) \in \mathcal{S} : d(s, t) \leq T\}$;
8 **end**
9 **return** $S$;

*3.1.4 Algorithm.* Having defined the estimator $\hat{y}(e)$, the path sampling procedure, and the probability distribution $f(p)$, we are ready to define the greedy approximation algorithm SAP.

Let $\Gamma = \max\{\text{out-deg}(s), \text{in-deg}(t) : (s, t) \in \mathcal{S}\}$, and let $L = 2|\mathcal{S}|^2\Gamma^2 \log(2m^3)n^{2T_0}$. At each iteration, $L$ determines the number of paths to be sampled by the path-sampling procedure with bias parameter $\gamma$. Once $L$ paths have been sampled, $\hat{y}(e)$ is computed for every edge in $G$ and the edge $e'$ with maximum $\hat{y}(e)$ value is selected to be included in the solution. The edge $e'$ is removed from $G$, $\mathcal{S}$ is updated to remove pairs that are already more than $T$ apart, and the process repeats until all pairs $(s, t) \in \mathcal{S}$ satisfy $d(s, t) > T$.

*3.1.5 Performance Guarantee.*

Theorem (Hoeffding's inequality [36]). *Suppose $Y_1, \ldots, Y_L$ are independent random variables in $[0, K]$. Let $Y = \frac{1}{L}\sum_{i=1}^{L} Y_i$. Given $t \geq 0$, the probability $\mathbf{P}\left(|Y - \mathbb{E}[Y]| \geq t\right) \leq 2\exp\left(\frac{-2Lt^2}{K^2}\right)$.*

Theorem 3.3. *With probability $1 - 1/m$, the solution $S$ returned by SAP satisfies $O(T_0 \log n) \cdot OPT \geq |S|$.*

Proof. In each iteration $i$ of the loop in line 2, let $\mathcal{P}_i$ be the paths in $\mathcal{P}$ still unbroken at this iteration, and let $\mathcal{S}_i, \Gamma_i$ have their assigned values during iteration $i$. Let $e \in E$. When $\hat{y}(e)$ is computed on line 5 of iteration $i$,

$$|\hat{y}(e) - y(e)| \geq \frac{|\mathcal{P}_i|}{2|\mathcal{S}_i|\Gamma_i} \tag{5}$$

with probability at most $\frac{1}{m^3}$; this holds by an application of Hoeffding's inequality since the path samples within each iteration of the while loop on line 2 are pairwise independent and the choice

of $L = 2|\mathcal{S}_i|^2\Gamma^2 \log(2m^3)n^{2T_0}$, and by the facts that the probability of any path in $Q$ is at least $1/n^{T_0}$ (so $K \leq n^{T_0}$ in Hoeffding's inequality) and $|\mathcal{P}_i| \leq n^{T_0}$. Hence the probability that (5) holds for any edge $e$ in any iteration is at most $1/m$ by the union bound. For the rest of the proof, we assume the negation of (5) for all edges and iterations (which happens with probability at least $1 - 1/m$).

Let $o_i \leq OPT$ be the minimum number of edges to break all paths in $\mathcal{P}_i$. In each iteration, there exists an edge $e \in E$ that lies upon at least $\frac{|\mathcal{P}_i|}{o_i}$ paths by the pigeon hole principle. Then, with $e'$ as chosen on line 6 of iteration $i$, $y(e') \geq \frac{|\mathcal{P}_i|}{o_i} - \frac{|\mathcal{P}_i|}{2|\mathcal{S}_i|\Gamma_i} \geq \frac{|\mathcal{P}_i|}{2o_i}$, since $|\mathcal{S}_i|\Gamma_i \geq o_i$ by the choice of $\Gamma_i$. Hence,

$$|\mathcal{P}_{i+1}| \leq \left(1 - \frac{1}{2o_i}\right)|\mathcal{P}_i| \leq \left(1 - \frac{1}{2OPT}\right)|\mathcal{P}_i| \leq \left(1 - \frac{1}{2OPT}\right)^{i+1}|\mathcal{P}|. \tag{6}$$

By (6), we can determine an iteration $g$ such that $|\mathcal{P}_g| \leq 1$. Using the facts that $|\mathcal{P}| \leq n^{T_0}$ and $\log(1+x) \geq x - \frac{x^2}{2}$ for $x \geq 0$, we have

$$g \leq \frac{\log|\mathcal{P}|}{\log\left(1 + \frac{1}{2OPT-1}\right)} \leq \frac{T_0 \log n(2OPT-1)}{1 - \frac{1}{2(2OPT-1)}} = O(T_0 \log n)OPT.$$

□

*3.1.6 Tight examples.* Notice that the proof of Theorem 3.3 proves a tighter ratio of $O(\log|\mathcal{P}|)$, on which $O(T_0 \log n)$ is an upper bound. Next, we construct a series of examples wherein SAP does pick a solution $S$ with $|S|/OPT \geq \Omega(\log|\mathcal{P}|)$, demonstrating that this analysis is tight up to a constant factor. At the beginning of the construction, $G$ contains two isolated nodes, $s, t$. Add nodes $g_1, \ldots, g_k$ and edges $(s, g_i)$ for each $g_i$. Next, add nodes $o_1, o_2$ to the graph, along with edges $(o_1, t), (o_2, t)$. Then, for each $g_i$, add vertices and edges to create $2^{i-1}$ edge-disjoint paths of length 2 between $g_i$ and $o_1$, and similarly create paths between $g_i$ and $o_2$. Let $d(u, v) = 1$ for all edges in $G$. Then SAP may during its sampling correctly estimate which edge lies on the most paths and would then select edges $(s, g_k), \ldots, (s, g_1)$ in that order, while the optimal solution is $\{(o_1, t), (o_2, t)\}$. So SAP picks $k$ edges when the optimal size is 2; since the number of paths is $2^{k+1}$, $|S|/OPT \geq \Omega(\log|\mathcal{P}|)$.

*3.1.7 Time complexity.* Each iteration of SAP conducts $L$ samples, runs Dijkstra's algorithm for each remaining pair, and adds a single edge to the solution $S$; we have a time complexity of $O(k(m + n\log n))$ per iteration for these computations. Since there are $|S|$ iterations, the overall complexity is $O\left(\left(n^{2T_0}k^2\Gamma^2 \log m + k(m + n\log n)\right)|S|\right)$. Furthermore, we remark that the quantity $k\Gamma$ may be replaced with any upper bound on OPT. Hence, if such a bound $U$ is obtained, the time complexity becomes $O\left(\left(n^{2T_0}U^2 \log m + k(m + n\log n)\right)|S|\right)$. In our implementation, we improve this further by sampling an equal number of paths $L^*$ per pair and parallelizing the sampling process by pair; for an upper bound $U$, we use the primal-dual solution. By returning the minimum of $U$ and the solution of SAP, our implementation therefore has an approximation ratio of $T_0$. Finally, in our implementation we set $L = U \log m$, which resulted in good empirical performance. Whenever possible, we speed up Dijkstra computations by using the $A^*$ algorithm [37], utilizing previous Dijkstra computations as an admissible heuristic.

## 3.2 Multicut Iterative Approach

In this section, we present another algorithm MIA (Alg. 2) to approximate LB MULTICUT with ratio $O(Mn^{11/23})$. In contrast to SAP, MIA is deterministic, and it is our only algorithm with a performance ratio that does not directly depend on $T_0$. In the worst case, MIA may require superpolynomial time (Section 3.2.4), but we found our implementation to be highly scalable in practice.

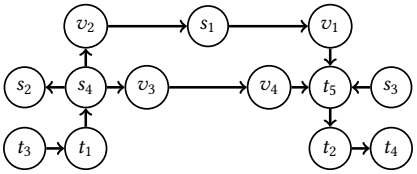



Fig. 1. $\mathcal{S} = \{(t_1, s_1), (s_1, t_2), (t_3, s_2), (s_3, t_4), (s_4, t_5)\}$; Every edge of $G$ lies upon a shortest path of length 3 between one of these pairs.

*3.2.1 Overview.* The general idea of MIA is as follows: until a feasible solution to the problem is produced, perform an approximate multicut on a subgraph of $G$, wherein the subgraph is constructed in order to ensure that the optimal multicut of $\mathcal{S}$ on this subgraph is a lower bound on the optimal solution. The solution produced is the union of these successive multicuts, and since each optimal multicut is a lower bound on OPT, the performance ratio is the number $M$ of multicuts performed multiplied by the performance ratio of the approximation algorithm used for multicut.

When $k > 1$, there are a number of difficulties in the application of the algorithm of Baier *et al.* [15], which is a $T/2$-approximation to the *single-pair* LB CUT problem. The algorithm of Baier *et al.* relies upon a minimum cut on the subgraph of shortest paths between $(s, t)$, and they show this cut lower bounds the optimal solution. First, the subproblem of computing a min cut becomes a multicut, which is already NP-hard [1]. More significantly, a multicut on the subgraph of shortest paths is no longer a lower bound on the optimal solution, as discussed below. Therefore, we introduce the notion of *path compatibility*, which ensures that a multicut of compatible paths maintains a lower bound on the optimal size of the length-bounded multicut. In addition, compatible paths are not required to be the same length, so it is possible that more paths are taken care of by a single multicut than if only shortest paths are considered.

*Counterexample.* Next, we show that a multicut on the graph induced by shortest paths does not lower bound the optimal solution. Consider the example shown in Fig. 1, with $T = 3$. Notice that $G$ is the subgraph formed by shortest paths (of length 3) between every pair in $\mathcal{S}$. However a min. multicut has size 4, while the min LB MULTICUT with $T = 3$ has size 3; the reason is that a longer path, which the bounded multicut does not have to cut, is created from $s_4$ to $t_5$ from pieces of shortest paths, namely the path $s_4 v_2 s_1 v_1 t_5$ of length 4, a situation that cannot occur in the case $|\mathcal{S}| = 1$, as shown in Baier *et al.* [15]. In order to obtain minimum multicuts that do lower bound the optimal solution, we introduce the notion of path compatibility. Sets of compatible paths do not allow the possibility of the creation of longer paths between pairs in $\mathcal{S}$.

*3.2.2 Path Compatibility in Directed Graphs.* Intuitively, for two paths to be compatible, we desire that no longer path between a pair in $\mathcal{S}$ can be created from the union of pieces of these two paths, as occured in Fig. 1 from the two shortest paths from $t_1$ to $s_1$ and from $s_1$ to $t_2$. To ensure longer paths are not created, if two paths $p$ and $q$ intersect, we require them to do so at the same distance along each path. Since the paths are directed, we cannot travel along part of $p$ and part of $q$ to obtain a longer path.

**Definition 1** (Compatible paths)**.** Let $p = p_0 \dots p_l$, $q = q_0 \dots q_s$ be simple paths in directed graph $G$. Then, $p$ and $q$ are *compatible* if for all $v \in q \cap p$, $v = p_i = q_j$ implies $\sum_{w=1}^{i} d(p_{w-1}, p_w) = \sum_{w=1}^{j} d(q_{w-1}, q_w)$.

In the following, we refer to the *distance along* a path $p$ to a node $v \in p$ as the sum of the edge weights on $p$ until $v$ is reached. Next, we formalize the intuition that longer paths cannot be created from a set of compatible paths in the following Lemma.

Lemma 3.4. *Let $\mathcal{Q} \subseteq \mathcal{P}(G, d, T)$ be a set of pairwise compatible paths. There does not exist a path $r$ in $G' = (V, E(\mathcal{Q}))$ between a terminal pair in $\mathcal{S}$ such that $d(r) > T$.*

Proof. Let $r$ be a path in $G'$ between a pair $(s, t) \in \mathcal{S}$. Since $r \in G'$, there must exist paths $p_1, \ldots, p_k \in \mathcal{Q}$, such that $r$ is composed of successive segments $r_i \subseteq p_i$. The last node in segment $r_i$ is the first node in $r_{i+1}$, label this node $v_i$. Then $v_1 \in p_1 \cap p_2$ and since these paths are compatible, $v_1$ is the same distance $d_1$ along $p_1$ and $p_2$ and hence is also distance at most $d_1$ along $r$. Furthermore, path $p_2$ is traversed from $v_1$ to $v_2$ and hence $v_2$ is at distance $d_2 > d_1$ along paths $p_2, p_3$, and at most this distance along $r$. Thus, the distance of $v_{k-1}$ along path $p_k$ is at most the distance of $v_{k-1}$ along $r$. Since the remainder of $r$ follows path $p_k$, the length of $r$ is at most the length of $p_k$, so $d(r) \leq d(p_k) \leq T$. □

Finally, we prove that the size of a multicut of a set of compatible paths, each of length at most $T$, is a lower bound on the optimal solution.

Lemma 3.5. *Let $\mathcal{Q} \subseteq \mathcal{P}$ be a set of pairwise compatible paths. Let $G' = (V, E(\mathcal{Q}))$ be the subgraph of $G$ formed by the edges of all paths in $\mathcal{Q}$. Then a minimum directed multicut of $\mathcal{S}$ on $G'$ is a lower bound on OPT, the minimum length-bounded multicut of $\mathcal{S}$.*

Proof. Since $G'$ is a subgraph of $G$, we have that $OPT(G', \mathcal{S}, T) \leq OPT(G, d, \mathcal{S}, T)$. By Lemma 3.4, every path $r$ in $G'$ between any pair in $\mathcal{S}$ has $d(r) \leq T$. Hence, to ensure each pair $(s, t) \in \mathcal{S}$ satisfies $d(s, t) > T$, it is necessary to cut all paths between $(s, t)$ in $G'$; so $d(s, t) = \infty$. Therefore, a minimum multicut of $\mathcal{S}$ on $G'$ has size $OPT(G', \mathcal{S}, T)$. □

**ALGORITHM 2:** MIA: **M**ulticut **I**terative **A**pproach

**Input** : Instance $(G, d, \mathcal{S}, T)$ of LB MULTICUT
**Output**: Solution $S \subseteq E$ to LB MULTICUT

$S = \emptyset$;
**while** $\min_{(s,t)\in\mathcal{S}} d_{G\setminus S}(s, t) = t \leq T$ **do**
  $\mathcal{P}_{short} = \{p \in \mathcal{P}(G\setminus S) : p$ is a shortest path between its endpoints $\}$;
  **while** $\mathcal{P}_{short} \neq \emptyset$ **do**
    $\mathcal{Q} = \emptyset$;
    **for** $p \in \mathcal{P}_{short}$ **do**
      **if** $p$ *is compatible with every* $q \in \mathcal{Q}$ **then**
        $\mathcal{Q} = \mathcal{Q} \cup \{p\}$;
      **end**
    **end**
    $G' = (V, E(\mathcal{Q}))$;
    $S = S \cup$ MIN. MULTICUT$(G', \mathcal{S})$;
    $\mathcal{P}_{short} = \mathcal{P}_{short} \setminus \mathcal{Q}$;
  **end**
**end**
**return** $S$;

*3.2.3 Algorithm.* The MIA algorithm (Alg. 2) proceeds as follows. First, we obtain all of the shortest paths in $\mathcal{P}$; these paths may not all have the same length since different terminals may be different distances apart. Next, a maximal set $\mathcal{Q}$ of pairwise compatible paths is constructed. Notice that $\mathcal{Q}$ depends on the order in which paths in $\mathcal{P}$ are considered. Finally, once $\mathcal{Q}$ is maximal, we construct $G' = (V, E(\mathcal{Q}))$ and perform a multicut on $(G', S)$; these edges are added to $S$. Since

**ALGORITHM 3:** TAG: Fully dynamic $T_0$-competitive algorithm

**Input** : Instance $(G, d, \mathcal{S}, T)$ of LB MULTICUT
**Output**: Solution $S \subseteq E$ to LB MULTICUT

```
1  Let S, W, P ⊆ E, 𝒰 ⊆ 𝒫 function h : (S ∪ W) → 𝒰;
2  S ← ∅, h ← ∅, P ← ∅, W ← ∅, 𝒰 ← ∅;
3  Run augment procedure (Section 3.3.1);
4  Run prune procedure (Section 3.3.2);
5  while change c to G arrives do
6  |  Ignore c if it is addition or removal of isolated vertex v;
7  |  if c is addition of edge e then
8  |  |  TAG-ADD(e);
9  |  end
10 |  if c is removal of edge e then
11 |  |  TAG-REMOVE(e);
12 |  end
13 end
```

multicut is NP-hard, we must apply an approximation algorithm for this problem – the best known algorithm has approximation factor $O(n^{11/23})$ [25]. These edges break all paths in $\mathcal{Q}$; finally, this process is repeated until all paths in $\mathcal{P}$ are broken.

Theorem 3.6. *Let $(G, d, \mathcal{S}, T)$ be an instance of* LB MULTICUT. *Then,* MIA *returns a feasible solution $S$ within a factor $O(Mn^{11/23})$ of $OPT(G, d, \mathcal{S}, T)$, in which $M$ is the number of multicuts required on line 12.*

Proof. Since every path in $\mathcal{P}$ must be present in some $\mathcal{Q}$, it is clear that $S$ is a feasible solution. Write $S$ as the disjoint union of approximate multicuts on line 12: $S = S_1 \dot\cup S_2 \dot\cup \ldots \dot\cup S_M$. Let $O_i$ be an optimal multicut corresponding to the multicut $S_i$. By Lemma 3.5, $|O_i| \leq OPT(G, d, \mathcal{S}, T)$ for each $i$. Furthermore, $|S_i| \leq O(n^{11/23})|O_i|$ by [25]. The result follows since $S$ is the disjoint union of the $S_i$'s. □

*3.2.4 Time complexity.* MIA requires $M$ multicuts; denote the approximation ratio of the multicut algorithm chosen as $r_{MC}$ and its running time as $t_{MC}$. In the unweighted case, compatibility of paths can be tested by using disjoint sets of vertices $V_0, \ldots, V_{T-1}$ corresponding to the paths in $\mathcal{Q}$ in the following way. For all $q = q_0 \ldots q_i \in Q$, place $q_j \in V_j$ for $j \in \{0, \ldots, i\}$. Then to check whether a new path $p = p_0 \ldots p_k$ is compatible with all paths in $\mathcal{Q}$, one may simply check if $p_i \in V_j$ for some $i \neq j$. If not, $p$ is compatible with all paths in $\mathcal{Q}$ and $p_i$ is added to $V_i$ for each $i$. In the worst case, this check requires $O(Tn)$ time. Hence, the time complexity of MIA on unweighted graphs is $O\left(M\left(\max\{|\mathcal{P}_{short}|\} \cdot Tn + t_{MC}\right)\right)$, where $\max\{|\mathcal{P}_{short}|\}$ is the maximum size of the sets of shortest paths between pairs in $\mathcal{S}$ considered in MIA. We implemented this unweighted version of the algorithm, using the approximation algorithm of Gupta *et al.* [24] for the multicut subproblems, which has a ratio of $O(\sqrt{n})$, leading to a performance ratio of $O(M\sqrt{n})$ for our implementation. Since listing all shortest paths in $\mathcal{P}_{short}$ may take superpolynomial time, MIA may require superpolynomial time; in practice, our implementation is shown to be highly scalable in Section 4.

### 3.3 Fully Dynamic $T_0$-Competitive Algorithm

In this section, we present our **T**$_0$-competitive **AlG**orithm TAG that is capable of dynamically updating its solution to LB MULTICUT upon incremental changes to the network. After each change

**ALGORITHM 4:** TAG-ADD($e$)

**Data:** Graph $G = (V, E)$, sets $S, W, P \subseteq E$, $\mathcal{U} \subseteq \mathcal{P}$, function $h : (S \cup W) \to \mathcal{U}$
**Input:** An edge $e$ to be added to $G$

$E \leftarrow E \cup \{e\}$ ;
Run augment procedure (Section 3.3.1);
Run prune procedure (Section 3.3.2);

**ALGORITHM 5:** TAG-REMOVE($e$)

**Data:** Graph $G = (V, E)$, sets $S, W, P \subseteq E$, $\mathcal{U} \subseteq \mathcal{P}$, function $h : (S \cup W) \to \mathcal{U}$
**Input:** An edge $e \in E$ to be deleted

$E \leftarrow E \backslash \{e\}$;
**if** $e \in S \cup W$ **then**
    $p \leftarrow h(e)$;
    $h \leftarrow h \backslash \{(f, p)\}, \forall f \in E(p)$;
    $S \leftarrow S \backslash E(p)$;
    $W \leftarrow W \backslash E(p)$;
    $\mathcal{U} \leftarrow \mathcal{U} \backslash \{p\}$;
**end**
Run augment procedure (Section 3.3.1);
Run prune procedure (Section 3.3.2);

to the graph, TAG ensures that its solution maintains a worst-case, performance ratio of $T_0$ to the optimal solution on the updated problem instance; that is, TAG has a *competitive ratio* of $T_0$.

In overview, TAG ensures that its solution $S$ lies within the union $U$ of the edges of a pairwise edge-disjoint collection of paths $\mathcal{U} \subseteq \mathcal{P}$; since these paths are pairwise edge-disjoint, a worst-case guarantee of $T_0$ is enforced on $U$ and therefore $S$ since $S \subseteq U$. Internally, TAG monitors a set $W = U \backslash S$ of *pruned edges* that is disjoint from $S$. To enable its solution to be efficiently updated, TAG maintains a function $h : S \dot{\cup} W \to \mathcal{U}$ that maps edges to the pairwise edge-disjoint set of paths $\mathcal{U}$.

Once a solution as described above is obtained, TAG maintains it by running TAG-ADD whenever an edge is added, and TAG-REMOVE whenever an edge is removed. To delete a vertex, all of its edges may be deleted one by one and then the vertex may be deleted from the graph. Vertex addition may be handled similarly, so we restrict our attention to edge insertion and removal in the rest of this section.

In order to maintain its solution $S$ with performance guarantee, TAG preserves four properties at all times.

$$h : S \cup W \to \mathcal{U} \text{ is a well-defined function} \tag{$T.1$}$$

$$\mathcal{P}(G \backslash S) = \emptyset \tag{$T.2$}$$

$$\mathcal{U} \text{ is a pairwise disjoint subset of } \mathcal{P} \tag{$T.3$}$$

$$S \cup W = \bigcup_{p \in \mathcal{U}} E(p) \tag{$T.4$}$$

Collectively, we refer to these four properties as the TAG *properties*. Property ($T.1$) is important for TAG-REMOVE to be able to update the solution, while properties ($T.2$) – ($T.4$) ensure the feasibility and performance guarantee for the solution $S$ to LB MULTICUT, as we show in the following lemma.

Lemma 3.7. *Let $G = (V, E)$, $S, W, h, \mathcal{U}$ satisfy the* TAG *properties. Let $O \subseteq E$ be an optimal solution to* LB MULTICUT *on $G$. Then $S$ is a feasible solution for* LB MULTICUT *and $|S| \leq T_0|O|$.*

Proof. First, we show $|\mathcal{U}| \leq |O|$. Since $O$ is a feasible solution to LB MULTICUT, there must be an element of $O$ in each path in $\mathcal{U}$. Since $\mathcal{U}$ is pairwise disjoint by (*T*.3), $|\mathcal{U}| \leq |O|$. Therefore, by properties (*T*.3) and (*T*.4,) $|S| \leq |S \cup W| \leq T_0|\mathcal{U}| \leq T_0|O|$, since each path $p \in \mathcal{U}$ has at most $T_0$ edges. Finally, (*T*.2) implies $S$ is a feasible solution to LB MULTICUT. □

All of the TAG algorithms (Algs. 3, 4, 5) rely upon two procedures important for the maintaince of the TAG properties, which we describe in the next two sections.

*3.3.1 Augment procedure.* This procedure ensures that the solution $S$ is feasible to the current problem instance; it works by first attempting to break a path $p \in \mathcal{P}(G \backslash S)$ by moving an edge from $W$ to $S$ if possible; if not, it adds all edges on $p$ to $S$ and $P$, and sets $h(f) = p$ for each edge $f \in E(p)$. This process repeats until the solution $S$ is feasible. The set $P \subseteq S$ comprises edges that could be potentially *pruned* from $S$ (i.e. moved from $S$ to $W$).

Next, we prove an important lemma for the augment procedure, namely that TAG properties (*T*.1), (*T*.3), and (*T*.4) are preserved and (*T*.2) is satisfied after the termination of the augment procedure.

Lemma 3.8. *Suppose the augment procedure begins with $S, W, h, \mathcal{U}$ initially satisfying properties (T.1), (T.3), and (T.4). Then, these three properties remain satisfied at the termination of augment, and (T.2) is satisfied as well.*

Proof. Moving an edge from $W$ to $S$ has no effect on any of the properties. If the definition of $h$ is extended, it had not been previously defined on any of these elements since they were not in $S \cup W$, so property (*T*.1) is preserved. Also, by property (*T*.4), when $h$ is extended, $p$ is disjoint from any triangles already in $\mathcal{U}$, ensuring property (*T*.3); furthermore, $E(p)$ is added to $S$, maintaining property (*T*.4). Finally, augment terminates only when $\min_{(s,t)\in\mathcal{S}} d_{G\backslash S}(s, t) > T$, which implies $\mathcal{P}(G\backslash S) = \emptyset$, which is (*T*.2). □

*3.3.2 Prune procedure.* This procedure examines each edge $e$ in $P$ in an arbitrary order and prunes it (moves $e$ from $S$ to $W$) if doing so does not cause $S$ to become infeasible, which requires a Dijkstra distance computation for each pair. After attempting to prune $e$, it removes $e$ from $P$, so at termination, $P = \emptyset$.

Lemma 3.9. *If the four* TAG *properties hold when the prune procedure is called, they continue to hold after it terminates.*

Proof. The prune procedure only moves edges from $S$ to $W$, so properties (*T*.1), (*T*.3), and (*T*.4) remain unaffected. Furthermore, it explicitly checks to make sure (*T*.2) is unaffected by the pruning of each edge. □

*3.3.3 The competitive ratio for* TAG. In this section, we prove the fully adaptive competitive ratio. First, we prove that TAG-ADD and TAG-REMOVE preserve the TAG properties.

Lemma 3.10. *Suppose $S, W, h, \mathcal{U}$ satisfy the* TAG *properties on $G = (V, E)$. Suppose an edge $e$ is added or removed from $G$. Then all four* TAG *properties are maintained after termination of* TAG-ADD($e$) *or* TAG-REMOVE($e$)*, respectively.*

Proof. For the case an edge $e$ is added to $E$, the only property potentially violated by this change is feasibility (*T*.2); by Lemmas 3.8 and 3.9, the call to the augment procedure thus ensures all four properties are satisfied and remain so after pruning.

Suppose an edge $e$ is removed from $E$. If $e \in S \cup W$, then $p = h(e)$ is defined, and $h$ is defined on all edges $E(p)$. TAG-REMOVE removes all of these edges from the domain of $h$, and it updates the set

$\mathcal{U}$ and the definition of $h$ to maintain properties ($T.1$), ($T.3$), and ($T.4$). However, feasibility may be violated when $E(p)$ is removed from $S$. If feasibility is violated, it is restored by calling the augment procedure on line 9 by Lemma 3.8. Thus, all four properties hold when the prune procedure begins, so the result follows from Lemma 3.9. □

Finally, we are ready to prove the fully adaptive performance ratio of TAG:

Theorem 3.11. *Suppose we have graph $G$ and a sequence of collections of edge additions or deletions $c_1, c_2, \ldots, c_l$, which produces a sequence of graphs $G = G_0, G_1, \ldots, G_l$. Then running* TAG *results in feasible solution $S_i$ to* LB MULTICUT *on each $G_i$. If $O_i$ is an optimal solution to* LB MULTICUT *on $G_i$, then $|S_i| \leq T_0|O_i|$.*

Proof. First, let $i = 0$. Before the call to the augment procedure from TAG, every TAG-property is satisfied with $S = W = h = \mathcal{U} = \emptyset$ except for possibly feasibility (property $T.2$). By Lemma 3.8 all four properties become satisfied. Hence, by Lemma 3.9, all four properties hold after the call to PRUNE. Thus, the hypotheses of Lemma 3.7 are satisfied on $G_0$.

Inductively assume the statement is true for $i-1$. Then by employing Lemma 3.10, the hypotheses of Lemma 3.7 remain satisfied for $S_i$ and $G_i$, which implies the result. □

*3.3.4 Time complexity.* The running time of TAG depends on the number of Dijkstra computations required. In our implementation of TAG, we first run Dijkstra on each pair in parallel. The augment procedure then proceeds sequentially to update this distance between each pair using the $A^*$ algorithm [37], with each update resulting in the removal of all edges on a shortest path between a pair. Hence, the number $K$ of edge-disjoint paths of length at most $T$ between every pair in $\mathcal{S}$ is an upper bound on the number of shortest-path computations required during augment. Finally, the prune procedure attempts to re-insert each edge in $P$ (of size at most $T_0OPT$) which requires a shortest-path computation for each pair – these computations again are sped up in practice with $A^*$ and are parallelized. Hence, the running time before any changes to the graph of TAG is $O\left(k\left(1 + K + T_0OPT\right)\left(m + n\log n\right)\right)$, where $U$ is any upper bound on the optimal size of a solution; the factor of $k$ can be removed if $k$ threads are available to parallelize the $A^*$ computations.

Next, we consider the time complexity of TAG-ADD. Suppose an edge $e$ is added to the graph; augment must recompute the distance between each pair in $\mathcal{S}$; if $e$ is chosen into $S$, augment will terminate. However, if $e$ is not added to $S$, it will require $k$ shortest path computations. If $e$ is added, then the prune procedure must attempt to prune each edge on a path with at most $T_0$ edges. Thus, the running time of TAG-ADD after a single edge addition is bounded by $O\left(k + \min\{m, T_0\}(m + n\log n)\right)$. In addition, we speed up the $k$ shortest-path computations of augment by only running Dijkstra from the tail of the inserted edge $e$ and terminating once there is no change in previously computed distances.

For TAG-REMOVE, consider the case a single edge $e$ is removed from the graph. If $e \notin S \cup W$, the time required is $O(1)$. Otherwise, TAG-REMOVE may add at most $T_0 - 1$ edges from $S$ back into the graph; in the worst-case, the augment procedure requires at most $\beta = \max\{|\mathcal{U}|, k, T_0 - 1\}$ shortest-path computations which may add at most $T_0 \cdot \beta$ edges to $P$. For each of these edges, the prune procedure requires the time of a single shortest-path computation, yielding a time complexity of $O\left(\min\{m, T_0\beta\}\left(m + n\log n\right)\right)$.

### 3.4 Discussion of proposed algorithms

In Table 2, we collect the results from the previous sections on the performance ratio and time complexity of each algorithm.

Recall that $U \leq m$ is an upper bound on OPT, $K \leq m$ is the number of edge-disjoint paths of length at most $T$ between every pair in $\mathcal{S}$, $r_{MC}$ and $t_{MC}$ are the ratio and time required, respectively,

Table 2. Algorithm comparison

| Algorithm | Performance Ratio | Time Complexity |
|---|---|---|
| SAP | $O(T_0 \log n)$ | $O\left(\left(n^{2T_0}U^2 \log m + k(m + n \log n)\right)|S|\right)$ |
| MIA | $O(Mr_{MC})$ | $O\left(M\left(\max\{|\mathcal{P}_{short}|\} \cdot Tn + t_{MC}\right)\right)$ |
| TAG | $T_0$ | $O\left(k(1 + K + T_0 OPT)(m + n \log n)\right)$ |

for the chosen algorithm for a directed multicut of $k$ pairs, and $\max |\mathcal{P}_{short}|$ is the maximum size of the sets of shortest paths between pairs in $\mathcal{S}$ considered in MIA.

*Performance ratios.* The worst-case performance ratio of TAG is strictly better than that of SAP; for a given problem instance and ordering of compatible paths chosen by MIA, the ratio of MIA may be better or worse than that of TAG. However, across all instances, the ratio of TAG is within a constant factor of the optimal performance ratio unless $NP \subseteq BPP$ as shown in Section 2.2.

*Running time.* If OPT is large, the number of samples required by SAP becomes large and its sampling procedure dominates its running time; this is ameliorated by trivially parallelizing the sampling process, which is possible since each sample is independent. In practice, the bias parameter $\gamma$ greatly reduces the number of samples required for a good solution; with $\gamma = 0.75$, we found that $O(U \log m)$ samples were sufficient to give good solutions in our experimental evaluations, where $U$ is an upper bound on OPT computed from a primal-dual solution. However, SAP must repeat the entire sampling process for each element chosen and hence scales linearly with the size of its solution.

Next, consider MIA. The multicuts performed by MIA are on subgraphs composed of edges from shortest paths in $\mathcal{P}_{short}$, and in our experimental evaluation these graphs were very sparse. Therefore, the multicuts proceeded quickly, despite the fact that the approximation algorithm of Gupta [24] requires a potentially expensive LP solution. In fact, we were even able to replace this LP with an exact IP solution of the multicut subproblems without changing the experimental running time very much; this substitution improves the performance ratio of MIA to simply $M$, the number of multicuts. However, since our implementation of MIA operates by listing all shortest paths in $\mathcal{S}$ and since there could be superpolynomial many of these paths, MIA may require superpolynomial time. In our evaluation, the memory requirement of listing all shortest paths was more restrictive than the time requirement on the networks we tested.

Finally, consider the non-incremental portion of TAG. The pruning procedure must compute the distance between each pair for each edge in its initial solution (at most $T_0 OPT$ edges). In practice, this pruning procedure dominates the running time of TAG, despite parallelization of the Dijkstra calculations by pair and speeding up the Dijkstra algorithm by using the $A^*$ algorithm.

## 4 EXPERIMENTAL EVALUATION

In this section, we evaluate all of our approximation algorithms and demonstrate that in practice they return nearly optimal solutions and scale to networks with billions of nodes and edges. In Section 4.2, we compare the algorithms on static instances of LB MULTICUT, while in Section 4.3 we evaluate the dynamic components of TAG. Finally, we summarize the key observations from the experimental results in Section 4.4.

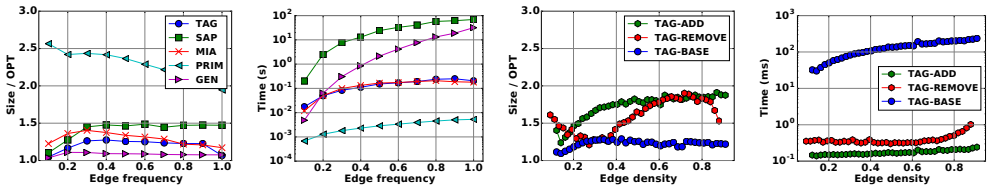
### 4.1 Methodology

We evaluated the following algorithms; the source code of all of our implementations is available [14].

- SAP (Alg. 1): we set $\gamma = 0.75$ unless otherwise noted and used number of samples $L = U \log m$, where $U$ is an upper bound on OPT obtained from the primal-dual algorithm; we found this value of $\gamma$ and number of samples to be sufficient to quickly return a good solution in most cases. By returning the minimum of the primal-dual solution $U$ and the solution found by SAP, the worst-case performance ratio of our implementation is improved to $T_0$. The number of samples $L$ is evenly divided among pairs in $\mathcal{S}$ and the sampling is parallelized by pair; further runtime improvement could be obtained by parallelizing the sampling process for a single pair. Finally, we re-use the Dijkstra hints from previous iterations until the number of valid paths in $\mathcal{P}$ sampled falls below a threshold; when this happens, we recompute the Dijkstra hints.
- MIA (Alg. 2): to solve the multicut subproblems, we used our implementation of Gupta's [24] $O\left(\sqrt{n}\right)$-approximation algorithm; to solve the directed multicut LPs therein, we used neither the ellipsoid method nor the equivalent polynomial-sized LPs [25], but rather a method that performs well in practice but may take exponential time: rather than using all paths between each pair in $\mathcal{S}$ as constraints, we iteratively add shortest LP-weighted paths as constraints until the length of the shortest such path between each pair exceeds 1.
- TAG (Alg. 3): all Dijkstra computations are sped up with $A^*$ method [37] and parallelized by pair in $\mathcal{S}$.
- PRIM: the primal-dual algorithm described in Section 2.
- GEN: the FPT-approximation algorithm from Kuhnle *et al.* [7] adapted to the edge version of the problem.
- OPT: the optimal solution obtained from an exact solution to IP 1 constructed by listing $\mathcal{P}$ and solved using IBM CPLEX Optimization Studio version 12.71.

GEN requires a listing of all $T$-bounded paths between each pair in $\mathcal{S}$ and hence was only able to run on small datasets. For SAP, We evaluated all algorithms in terms of the number of edges in the solution $S$ and the running time. To obtain target sets $\mathcal{S}$, we sampled uniformly random sets of pairs from each network, following Kuhnle *et al.* [7]. Unless otherwise stated, all results are averaged over 5 independent repetitions of each experiment.

The algorithms were evaluated on both synthesized networks and traces of real networks. The synthesized networks we considered were small Erdos-Renyi (ER) graphs (with $n = 100$), in which each potential edge has probability $p$ of appearing in the graph. On these small networks, we compare all algorithms with the optimal solution OPT. For real-world traces, we considered both weighted and unweighted topologies drawn from a variety of application domains. The real-world traces are listed in Table 3; with the exception of RoadSF, all of these networks were collected by the Stanford Network Analysis Project [8]. Gnutella is a peer-to-peer network; Enron is an e-mail communication network, Google is a world-wide-web graph, Skitter is an autonomous-system graph obtained from the internet, and Friendster is a social network. All of these networks are unweighted except for Skitter, which we weighted with a uniformly chosen weight in the interval $[1, 10]$ to simulate a QoS metric, following [38, 39]. Finally, RoadSF is the weighted, road network of San Francisco [40], which is normalized so that all distances lie in $[0, 10000]$. In addition, Table 3 lists an estimate $\hat{d}(x, y)$ of the average shortest-path distance in each network, obtained by sampling 10000 random, connected pairs and averaging their shortest-path distances. All of these topologies are undirected and were input to the algorithms in an adjacency-list format.




(a) Solution size, Static (b) Time (s), Static (c) Solution Size, Dynamic (d) Time (ms), Dynamic

Fig. 2. Results on small ER network ($n = 100$), with $T = 3, k = 10$

All experiments on datasets other than Friendster were run on a server with two Intel(R) Xeon(R) CPU E5-2667 @ 2.90GHz (12 cores) and with 256 GB RAM. The Friendster experiments were run on a server with Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30 GHz and 384 GB RAM. All algorithms were compiled with the GNU C++ compiler with optimization flag $-O3$ and allowed to use a maximum of 12 threads, and running time is reported as elapsed real-world time rather than CPU time.

Table 3. Real-world traces

| Network | $\|V\|$ | $\|E\|$ | $\hat{d}(x, y)$ | Weighted |
|---|---|---|---|---|
| Gnutella | $6.301 \times 10^3$ | $2.078 \times 10^4$ | 4.63 | No |
| Enron | $3.669 \times 10^4$ | $1.838 \times 10^5$ | 4.04 | No |
| RoadSF | $1.748 \times 10^5$ | $2.218 \times 10^5$ | 3635.29 | Yes |
| Google | $8.757 \times 10^5$ | $4.322 \times 10^6$ | 6.33 | No |
| Skitter | $1.696 \times 10^6$ | $1.109 \times 10^7$ | 13.00 | Yes |
| Friendster | $1.248 \times 10^8$ | $1.806 \times 10^9$ | 4.99 | No |

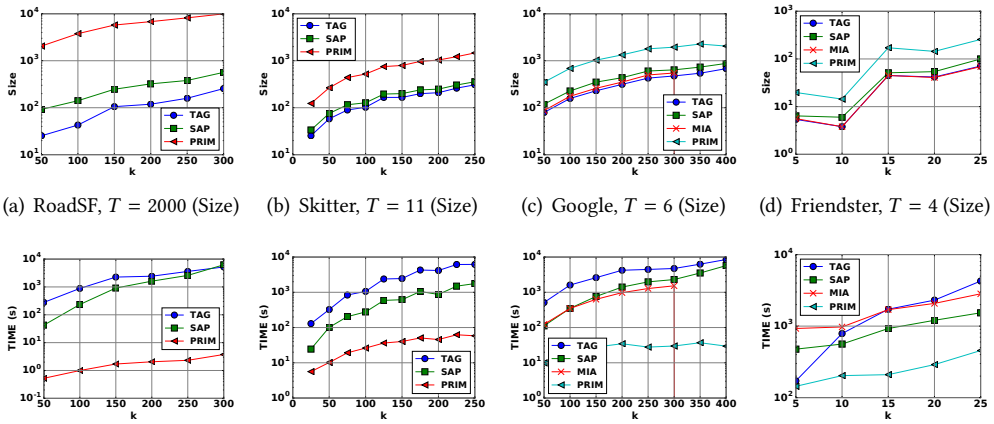
### 4.2 Static evaluation

In this section, we evaluate our algorithms on static graphs. TAG is terminated before it begins waiting for changes to the graph; we refer to running TAG this way as *static* TAG.

*4.2.1 Comparison to OPT.* In this set of experiments, we compare all algorithms to the optimal solution on unweighted ER networks with $n = 100$ and various edges densities. The threshold $T$ was set to 3, and the size of $\mathcal{S}$ was 10 pairs.

The solution size normalized by the size of OPT is shown in Fig. 2(a). First, notice that all algorithms besides PRIM remain under $1.5 \cdot OPT$ at all edge densities. For most edge densities ($p = 0.2$ to 0.7), the ranking from best to worst is GEN, TAG, MIA, SAP and PRIM. All algorithms except PRIM, MIA performed their individual best at the smallest edge density $p = 0.1$, which is promising since many real-world networks are sparse. At this edge density, SAP and TAG are within $1.1 \cdot$ OPT while MIA is within a factor of 1.20 from optimal. Although $T = 3$ is a favorable setting for PRIM, it performs worse than a factor of 2 from optimal in all cases, justifying the need for scalable, approximation algorithms for LB MULTICUT that perform well in practice.

In Fig. 2(b), we plot the average running time of each algorithm versus the edge density. In this experiment, we see how each algorithm scales with edge density when $n$ is fixed. PRIM is the fastest, and PRIM, MIA, and TAG each increase by roughly a factor of 10 as the edge density goes to 1.0, while GEN increases by a factor of roughly 1000. Surprisingly, SAP is the slowest algorithm in this




(a) RoadSF, $T = 2000$ (Size) (b) Skitter, $T = 11$ (Size) (c) Google, $T = 6$ (Size) (d) Friendster, $T = 4$ (Size)

(e) RoadSF, $T = 2000$ (Time) (f) Skitter, $T = 11$ (Time) (g) Google, $T = 6$ (Time) (h) Friendster, $T = 4$ (Time)

Fig. 3. Average results over 5 choices of $\mathcal{S}$ versus the number of pairs $k$ in $\mathcal{S}$. *Top row*: Size of solution versus $k$ on each dataset. *Bottom row*: Running time versus $k$ on each dataset.

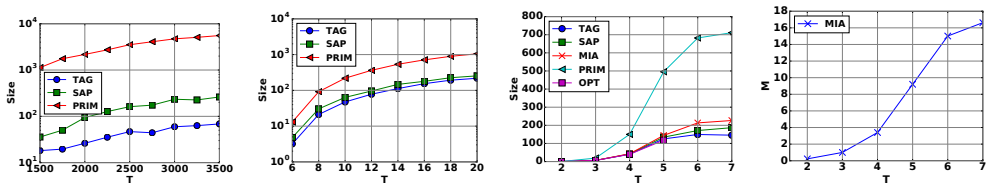
set of experiments and scales poorly with edge density. This results from it using the relatively large PRIM solution to compute its number of samples $L$ as discussed in Section 3.1.7. Despite this result, we will demonstrate in Section 4.2.2 that SAP scales to networks with billions of edges, while the FPT-approximation algorithm GEN is unable to run on our real-world traces.

*4.2.2 Results on large-scale, real networks.* In this section, we evaluate our algorithms on the large-scale, real-world traces. In Section 4.2.2, we present the results as the size of the target set increases and examine the effect of varying the threshold $T$ on the weighted networks RoadSF and Skitter. The FPT-approximation algorithm GEN was unable to run on any of these networks as the path listing procedure exceeded our memory limit for these networks and threshold values.

*Scalability with size of target set.* In this set of experiments, we evaluated the algorithms on our largest networks RoadSF, Google, Skitter, and Friendster, with threshold $T = 2000, 6, 11$, and 4, respectively. The results are shown in Fig. 3.

First, we discuss the results on our largest network, Friendster, in which $k$ was varied from 5 to 25. The size of the solution of each algorithm is shown in Fig. 3(d), while the running time is shown in Fig. 3(h). TAG and MIA return the best solution sizes; indeed, these algorithms were virtually indistinguishable in solution quality. Larger values of $k$ only tripled the running time of MIA, while the running time of TAG increased by a factor of 20; for the lower values of $k$, TAG was faster than MIA, but this flipped at $k = 15$. The next best algorithm in terms of solution quality was SAP, which stayed within a factor of 1.5 from MIA, TAG. Of these three, SAP was the fastest after $k = 10$ and always finished in under an hour. Finally, PRIM was consistently roughly a factor of 4 worse than the other algorithms in terms of solution quality, although it was by far the fastest.

On the Google network, we varied $k$ from 50 to 400; the algorithm evaluation is similar to that of the Friendster network. Notably, MIA exceeded 256 GB of RAM and so was unable to run after $k = 300$; this resulted from MIA needing to store all of the shortest paths between each pair, while the other algorithms only store one such path per pair. Also, in contrast to Friendster on which SAP was the fastest of the three, MIA was the fastest of the three when it could run, due to the larger values of $k$. Again, PRIM was the fastest overall but had by far the worst quality of solution.




(a) RoadSF, $k = 50$ (Size) (b) Skitter, $k = 50$ (Size) (c) Gnutella, $k = 50$ (Size) (d) Gnutella, $k = 50$ (Number of multicuts $M$)

Fig. 4. Average results over 5 choices of $\mathcal{S}$ versus the threshold $T$.

Table 4. Dynamic evaluation, real-world traces

| | Parameters | | | | STATIC ADD/REMOVE | | DYNAMIC ADD/REMOVE | | |
|---|---|---|---|---|---|---|---|---|---|
| Dataset | $m$ | $k$ | $T$ | $N_{seq}$ | Time (s) | Size | Time (ms) | SpeedUp $\cdot 10^3$ | Loss |
| Gnutella | $2.078 \cdot 10^4$ | 100 | 4 | $5.0 \cdot 10^3$ | 0.61 / 0.18 | 125.4 / 30.0 | 1.0 / 1.0 | 0.6 / 0.2 | 6.7% / 0.0% |
| Enron | $1.838 \cdot 10^5$ | 100 | 4 | $5.0 \cdot 10^4$ | 21.6 / 3.9 | 269.8 / 76.2 | 3.5 / 0.7 | 6.1 / 5.5 | 7.0% / 15.5% |
| Google | $4.322 \cdot 10^6$ | 100 | 6 | $1.0 \cdot 10^6$ | 3765.9 / 696.4 | 617.0 / 155.4 | 110 / 12 | 33.7 / 57.2 | 20.3% / 13.3% |
| Friendster | $1.806 \cdot 10^9$ | 10 | 5 | $1.0 \cdot 10^8$ | 38127 / 32465 | 94.0 / 49.6 | 1100 / 0.2 | 40 / 1.5 $\cdot 10^5$ | 0.0% / 2.8% |

Since our implementation of MIA is only for unweighted networks, on Skitter and RoadSF we dropped it from the evaluations. On the autonomous system graph Skitter with the simulated, QoS metric, the comparison is similar to the previous cases, with SAP slightly worse than TAG but SAP is almost a factor of 10 faster than TAG. With the larger threshold, PRIM has gotten worse with a solution quality of almost a factor of 5 higher than the other two.

Finally, we consider the road network RoadSF, on which the results deviate from those on the other networks in a couple of signficant ways. First, notice that PRIM is now almost a factor of 1000 worse than TAG. Next, RoadSF was the only network in which SAP struggled to obtain valid path samples (*i.e.* path samples in $\mathcal{P}$), since the number of hops on valid paths were so high. For this network, we set the $\gamma$ parameter of SAP to 0.99, whereas on other networks there were no problems with $\gamma = 0.75$. Recall that $\gamma$ controls how biased the sampling is with respect to shortest paths. With this compensation, SAP was able to obtain valid samples, but its solution quality is up to a factor of 5 worse than TAG, in sharp contrast to its performance on the other networks. Furthermore, its running time was close to that of TAG on this network.

*Scalability with threshold $T$.* In this section, we evaluated the performance of each algorithm as the threshold $T$ varies; we used the weighted networks RoadSF and Skitter for this evaluation, as well as the unweighted Gnutella network; since MIA is only implemented on unweighted networks, it is evaluted only on Gnutella. On all networks, we used target sets with $k = 50$.

Results are shown in Fig. 4 (running time not shown). On the RoadSF network, we varied $T$ from 1500 to 3500. As discussed in the previous section, SAP used $\gamma = 0.99$ here, and its solution quality is significantly worse than TAG, which performed the best. PRIM returned solutions roughly 1000 times larger than the corresponding solution of TAG; however, PRIM continued to run by far the fastest. The running time of TAG is the slowest and increases by a factor of more than 10 as $T$ varied.

On the Skitter network, we varied $T$ from 6 to 20. The solution qualities of TAG, SAP are much closer than on RoadSF, with TAG only slightly better; however SAP runs up to 10 times faster than TAG. PRIM maintained a solution quality of roughly 5 times worse than TAG as $T$ varied.

On the Gnutella network, the solution size for all four algorithms is shown in Fig. 4(c); we also plot the optimal solution size OPT until the path listing procedure ran out of memory. In Fig. 4(d), we plot the number $M$ of directed multicuts in MIA versus $T$; interestingly, the worst-case performance ratio $O(Mn^{11/23})$ of MIA increases rapidly with $T$ as the number of multicuts increases to more than 16; however, the performance ratio in practice of MIA to OPT does worsen with $T$ but stays within 2 on this network. This behavior is also exhibited for the other algorithms, except for PRIM, whose worst-case ratio is $T$; PRIM performs close to this worst case.

### 4.3 Dynamic evaluation

In this section, we evaluate the dynamic portion of TAG as compared with rerunning static TAG after all changes to the graph and the static optimal solution on small graphs.

*4.3.1 Comparison to OPT.* First, we compared TAG to the optimal solution on small ER graphs, with $n = 100$. To evaluate TAG under edge addition, we started from an ER graph with $p = 0.1$, then we added a sequence of $N_{seq}$ random edges to the graph to obtain a random graph with a new edge density. The original solution of static TAG at $p = 0.1$ is updated with TAG-ADD after each edge addition. To evaluate under edge removal, a similar procedure is employed, except that we start at an ER graph with $p = 0.9$ and remove a sequence of $N_{seq}$ edges, running TAG-REMOVE after each removal.

In Fig 2(c), we show the solution size versus edge density for each algorithm after using the above procedures for different values of $N_{seq}$. We observe that although static TAG is clearly superior, both TAG-ADD and TAG-REMOVE remain within a factor of 2 from optimal even after drastic changes in density. Furthermore, in Fig. 2(d), we plot the average running time for TAG-ADD and TAG-REMOVE over each sequence of $N_{seq}$ changes to the graph, and we compare with the running time of static TAG at that edge density. Notice that even on these small ER graphs TAG-ADD and TAG-REMOVE enjoy a factor 100 to 1000 speedup over static TAG .

*4.3.2 On large real-world traces.* Next, we evaluated the dynamic performance on the real-world traces. A similar procedure to the one on ER networks was employed; starting from the original network and static TAG solution, we added or removed a sequence of $N_{seq}$ edges from the network, running TAG-ADD or TAG-REMOVE to update the solution after change. For each network, $N_{seq}$ was chosen large enough to cause significant changes to the solution; except on the Friendster network, we used the same value of $N_{seq}$ for the addition and removal experiments.

The results are shown in Table 4. In the columns for STATIC ADD/REMOVE we show the average running times and sizes of static TAG after adding / removing a sequence of $N_{seq}$ edges from the graph. In the DYNAMIC ADD/REMOVE columns, we show that average running time over each change in each sequence for TAG-ADD / TAG-REMOVE, the average, final size of the solution after running TAG-ADD or TAG-REMOVE on all changes, the average factor of speedup over static TAG, and the percentage of solution quality lost as compared with static TAG after all changes.

Notice that for the smallest network, Gnutella, TAG-ADD and TAG-REMOVE resulted in speedups of 600 and 200, respectively, while only incurring a small loss from the static TAG solutions. On the larger Enron network, the speedups for TAG-ADD and TAG-REMOVE were an order of magnitude higher, at 6100 and 5500, respectively, while maintaining small loss percentages. On the Friendster network, we set $N_{seq} = 10^6$ for edge addition and $N_{seq} = 10^8$ for the removal experiment, since the average time for an edge addition exceeded a second, while the average time for an edge removal was less than a millisecond. We observed average speedups of 40,000 and $1.5 \times 10^8$ for addition and removal, respectively. The reason that TAG-REMOVE outperforms TAG-ADD on larger networks is TAG-ADD must run Dijkstra after every edge addition to see if the solution has become infeasible (see Section 3.3.4), while TAG-REMOVE safely ignores any edges removed that are not within $S \cup W$.

### 4.4 Summary of results

In this section, we summarize the experimental results.

- Of our three algorithms, the solution qualities of MIA and TAG on our real-world traces were very similar; in terms of running time, MIA scales better with $k$, but in terms of memory usage, TAG scales better with $k$. The solution quality of SAP was worse than the other two, but usually by only a small factor. In addition, SAP usually ran the fastest, although for high $k$ values it was occasionally beaten by MIA .
- As compared with static TAG, dynamic TAG demonstrated massive improvements in average running time of up to $10^8$ for edge removal and $10^4$ for edge addition and only lost small percentages of solution quality over millions of edge insertions and removals.
- The primal-dual algorithm PRIM consistently ran much faster than SAP, MIA, and static TAG, often by a factor of more than 10. However, its solution quality was usually worse by a factor of at least 5; on RoadSF, PRIM returned solutions nearly 1000 times larger than TAG.
- The FPT-approximation algorithm GEN performed well on the small ER networks with $n = 100$ but was unable to scale to larger networks due to both time and memory constraints stemming from the listing of all bounded paths required.
- On networks with very long path distances, SAP requires the parameter $\gamma$ to be set higher than 0.75 to obtain valid path samples. We observed this effect only on RoadSF, which had large distances corresponding to thousands of hops required for the sampling procedure.

## 5 CONCLUSIONS

In this work, we have presented three algorithms SAP, TAG, and MIA for the LB MULTICUT problem, each of which scales to networks with billions of edges and nodes in under a few hours and has a proven performance guarantee. In addition, our fully dynamic TAG enables large speedups over the static solution of TAG. Finally, we have shown that unless $NP \subseteq BPP$, there is no polynomial-time algorithm with ratio better than $\lfloor T/6 \rfloor - 1 - \epsilon$.

Future work would include lowering the number of samples required by SAP, thereby making it even more scalable. In addition, our definition of path compatibility only works for directed graphs. Extending this definition to undirected graphs would allow a version of MIA to employ an undirected multicut approximation, which have better worst-case guarantees. Finally, it is an interesting open question whether a polynomial-time algorithm for LB MULTICUT could have a better performance ratio than $T$, even in the special case when $T$ is a fixed parameter. We conjecture our inapproximability result could be improved to $T - \epsilon$.

## 6 ACKNOWLEDGEMENTS

The anonymous reviewers and Thang N. Dinh provided many helpful comments which improved the manuscript. This work was supported in part by NSF grants CNS-1443905 and EFRI 1441231, and DTRA grant HDTRA1-14-1-0055.

## REFERENCES

[1] Vijay V Vazirani. *Approximation Algorithms*. Springer-Verlag Berlin Heidelberg, first edition, 2003.
[2] Tony H Grubesic, Timothy C Matisziw, Alan T Murray, and Diane Snediker. Comparative Approaches for Assessing Network Vulnerability. *International Regional Science Review*, 31(1):88–112, 2008.
[3] Arunabha Sen, Sudheendra Murthy, and Sujogya Banerjee. Region-based connectivity - A new paradigm for design of fault-tolerant networks. In *2009 International Conference on High Performance Switching and Routing, HPSR 2009*, 2009.
[4] Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, 2014.
[5] Linus Thrybom and Gunnar Prytz. QoS in Switched Industrial Ethernet. *IEEE Conference on Emerging Technologies and Factory Automation*, 2009.

[6] Amazon.com. Amazon.com Help: Guaranteed Delivery Terms and Conditions. https://www.amazon.com/gp/help/customer/display.html?ie=UTF8&nodeId=201910260. Accessed: 2017-10-01.

[7] A. Kuhnle, T. Pan, V.G. Crawford, M.A. Alim, and M.T. Thai. Pseudo-separation for assessment of structural vulnerability of a network. In *SIGMETRICS 2017 Abstracts - Proceedings of the 2017 ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems*, 2017.

[8] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data. Accessed: 2017-10-01.

[9] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 177–187, 2005.

[10] Chunsheng Zhu, Lei Shu, Takahiro Hara, Lei Wang, Shojiro Nishio, and Laurence T. Yang. A survey on communication and data management issues in mobile sensor networks. *Wireless Communications and Mobile Computing*, 14:19–36, 2014.

[11] Qingchen Zhang, Chunsheng Zhu, Laurence T Yang, Zhikui Chen, Liang Zhao, and Peng Li. An Incremental CFS Algorithm for Clustering Large Data in Industrial Internet of Things. *IEEE Transactions on Industrial Informatics*, 13(3):1193–1201, 2017.

[12] Euiwoong Lee. Improved Hardness for Cut, Interdiction, and Firefighter Problems. *Arxiv preprint arxiv:1607.05133v1*, 2016.

[13] Michael Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, second edition, 2006.

[14] Alan Kuhnle. Source code link. https://gitlab.com/kuhnle/multi-pcut.

[15] Georg Baier, Thomas Erlebach, Alexander Hall, Ekkehard Koehler, Petr Kolman, Ondrej Pangrac, Heiko Schilling, and Martin Skutella. Length-Bounded Cuts and Flows. *ACM Transactions on Algorithms*, 7(1):1–27, 2010.

[16] Petr A. Golovach and Dimitrios M. Thilikos. Paths of bounded length and their cuts: Parameterized complexity and algorithms. *Discrete Optimization*, 8(1):72–86, 2011.

[17] Pavel Dvorak and Dusan Knop. Parameterized Complexity of Length-Bounded Cuts and Multi-cuts. In *Theory and Applications of Models of Computation: 12th Annual Conference*, pages 441—-452. Springer International Publishing, 2015.

[18] Dániel Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.

[19] K. Malik, A. K. Mittal, and S. K. Gupta. The k most vital arcs in the shortest path problem. *Operations Research Letters*, 8(4):223–227, 1989.

[20] Eitan Israeli and R. Kevin Wood. Shortest-Path Network Interdiction. *Networks*, 40(2):97–111, 2002.

[21] Gerald Brown, Matthew Carlyle, Javier Salmerón, and Kevin Wood. Defending critical infrastructure. *Interfaces*, 36(6):530–544, 2006.

[22] Justin Yates and Irene Casas. Role of Spatial Data in the Protection of Critical Infrastructure and Homeland Defense. *Applied Spatial Analysis and Policy*, 5(1):1–23, 2012.

[23] Naveen Garg, Vijay V Vaziranit, and Mihalis Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. In *Proceedings of the twenty-fifth annual ACM Symposium on Theory of Computing.*, pages 698–707, 1993.

[24] A Gupta. Improved results for directed multicut. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 454–455, 2003.

[25] Amit Agarwal, Noga Alon, and Moses S Charikar. Improved approximation for directed cut problems. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of Computing*, pages 671–680, New York, NY USA, 2007. ACM.

[26] Ben Roberts and Dirk P. Kroese. Estimating the Number of s-t Paths in a Graph. *Journal of Graph Algorithms and Applications*, 11(1):195–214, 2007.

[27] Peng Jun Wan, Ding Zhu Du, Panos Pardalos, and Weili Wu. Greedy approximations for minimum submodular cover with submodular cost. *Computational Optimization and Applications*, 45(2):463–474, 2010.

[28] T Soma and Y Yoshida. A Generalization of Submodular Cover via the Diminishing Return Property on the Integer Lattice. *Advances in Neural Information Processing …*, pages 1–9, 2015.

[29] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[30] Ahmad-Reza Sadeghi, Christian Wachsmann, and Michael Waidner. Security and Privacy Challenges in Industrial Internet of Things. *Proceedings of the 52nd Annual Design Automation Conference on - DAC '15*, 17:1–6, 2015.

[31] Inc. Institute of Electrical and Electronics Engineers. IEEE 802.1 Time-Sensitive Networking Task Group.

[32] Kaixin Xu Kaixin Xu, Xiaoyan Hong Xiaoyan Hong, Mario Gerla Mario Gerla, Henry Ly, D.L. Daniel Lihui Gu, and Los Angeles. Landmark routing in large wireless battlefield networks using UAVs. *2001 MILCOM Proceedings Communications for Network-Centric Operations: Creating the Information Force (Cat. No.01CH37277)*, 1(c):230–234, 2001.

[33] Syed R Ali and Richard S Wexler. Army Warfighter Information Network-Tactical ( Win-T ) Theory of Operation. In *IEEE Military Communications Conference (MILCOM)*. IEEE, 2013.

[34] Juan C. Juarez, Anurag Dwivedi, a. Roger Hammons, Steven D. Jones, Vijitha Weerackody, and Robert a. Nichols. Free-space optical communications for next-generation military networks. *IEEE Communications Magazine*, 44(November):46–51, 2006.

[35] Venkatesan Guruswami and Euiwoong Lee. Inapproximability of Feedback Vertex Set for Bounded Length Cycles. In *Electronic Colloquium on Computation Complexity (ECCC)*, volume 21, page 2, 2014.

[36] Wassily Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[37] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics*, 4(2):100–107, 1968.

[38] Guoliang Xue, Arunabha Sen, Weiyi Zhang, Jian Tang, and Krishnaiya Thulasiraman. Finding a path subject to many additive QoS constraints. *IEEE/ACM Transactions on Networking*, 15(1):201–211, 2007.

[39] Ying Xuan, Yilin Shen, Nam P. Nguyen, and My T. Thai. A graph-theoretic QoS-aware vulnerability assessment for network topologies. *GLOBECOM - IEEE Global Telecommunications Conference*, 2010.

[40] Thomas Brinkhoff. Generating Network-Based Moving Objects. In *IEEE 12th International Conference on Scientific and Statistical Database Management*, pages 8–10, 2000.

*Received November 2017, revised January 2018, accepted March 2018.*