Continuous Length-Bounded Paths Interdiction

Raed Alharbi*[1], Lan N. Nguyen*[2], and My T. Thai[2]

[1] Saudi Electronic University, Riyadh, 11673, Saudi Arabia,
`ri.alharbi@seu.edu.sa`
[2] University of Florida, Gainesville, FL, 32611,
{`lan.nguyen, mythai`}`@ufl.edu`

Abstract. Network vulnerability assessment, in which a communication between nodes is functional if their distance under a given metric is lower than a pre-defined threshold, has received significant attention recently. However, those works only focused on discrete domain while many practical applications require us to investigate in the continuous domain. Motivated by this observation, we study a **Length-bounded Paths Interdiction in Continuous Domain** (`cLPI`) problem: given a network $G = (V, E)$, in which each edge $e \in E$ is associated with a function $f_e(x)$ in continuous domain, and a set of target pairs of nodes, find a distribution $\mathbf{x} : E \to \mathbb{R}^{\geq}$ with minimum $\sum_{e \in E} \mathbf{x}(e)$ that ensures any path p, connecting a target pair, satisfies $\sum_{e \in p} f_e(\mathbf{x}(e)) \geq T$. We first propose a general framework to solve `cLPI` by designing two oracles, namely *Threshold Blocking* (`TB`) oracle and *Critical Path Listing* (`CPL`) oracle, which communicate back and forth to construct a feasible solution with theoretical performance guarantees. Based on this framework, we propose a bicriteria approximation algorithm to `cLPI`. This bicriteria guarantee allows us to control the solutions's trade-off between the running time and the performance accuracy.

1 Introduction

Components of a network never have the same important level. There always exists a group of nodes or edges which plays more critical role than the others on determining networks' performance. Literature has spent significant effort on studying and identifying such group both theoretically and practically. The very first efforts mostly were invested for the connectivity metric, in which a connection between two nodes is functional if there exists a path connecting them. This metric could be found in the Multicut problem [3,5,12,13], Multiway problem [22], or Graph Partitioning [2,15].

However, as modern networks are evolving, connectivity is no longer sufficient on guaranteeing networks' functionality or quality of services. Instead of removing, a slight change on components' behavior can downgrade the whole system's performance. For example, a congestion or traffic jams [6,7] on some roads can damage a delivery business; or a change on priority level [1] of packet

* Equal contribution

types on some routers can significantly delay communication between end systems, downgrading their quality of services.

Motivated by these observations, many recent researches turn the attention on network malfunction without damaging connectivity. For example, Kuhnle et al. [17] studied the problem of `LB-MULTICUT`: given a weighted network, a set of pairs of nodes and a threshold T, their work aims to identify a minimum set of edges whose removal cause the distance between a pair exceed T. By discarding the "remove" flavour, Nguyen et al. [20] extended this concept to introduce `QoSD` problem, in which an edge weight can be varied with an amount of efforts and the problem asks for a minimum amount of efforts for the same objective as in `LB-MULTICUT`. Other works can be found in [8–10, 18]. However, those works share the same trait that they are all discrete problems, thereby leaving the continuous domain largely opened.

Indeed, many applications require us to investigate the above problem in the continuous domain. For example, in information and telecommunication engineering, a channel capacity in communication network is theoretically measured by the signal-to-interference-plus-noise ratio (SINR) [14] with wireless communication and signal-to-noise ratio (SNR) [16] with wired channel. Such measurements are related to the power of the interfering signal and noises, which consists of continuous variables. The information transfer between two systems, thus, is determined by the delays on propagation channels, which can be modified by those variables. Another example can be seen in diffusion protocol [11] in Bitcoin P2P network, in which a node u relays a message to its neighbors with an independent, exponential delay rate λ_u. Increasing some values of λ_us can delay the packet propagation between major miners, damaging the network consensus.

Motivated by these applications, in this paper, we extend the `QoSD` problem into continuous domain by introducing the `cLPI` problem as follows: Given a directed network $G = (V, E)$, a set S of target pairs of nodes and a threshold T, an edge $e \in E$ is associated with a continuous and monotone increasing function $f_e : \mathbb{R}^{\geq} \to \mathbb{R}^{\geq}$, the `cLPI` problem asks for a distribution $\mathbf{x} : E \to \mathbb{R}^{\geq}$ with minimum $\sum_{e \in E} \mathbf{x}(e)$ such that any path p, connecting a pair in S, satisfies $\sum_{e \in p} f_e(\mathbf{x}(e)) \geq T$. For simplicity, we write $\mathbf{x}$ under a vector form $\{x_e\}_{e \in E}$ where $x_e = \mathbf{x}(e)$, thus $\sum_{e \in E} \mathbf{x}(e) = \|\mathbf{x}\|$ (Again for simplicity, we use notation $\|\cdot\|$ to indicate norm 1 of a vector). Another presentation of `cLPI`'s objective is to find $\mathbf{x}$ with minimum $\|\mathbf{x}\|$ that guarantees there exists no T-length-bounded multi-commodity flow on G. A T-length-bounded multi-commodity flow is a flow between the target pairs that can be decomposed into flow paths of length at most T. The solution $\mathbf{x}$ of `cLPI` can be used as a metric to measure the network's functionality: large $\|\mathbf{x}\|$ indicates the network is resilient to external interference or noises and able to maintain quality of service under extreme environment. Furthermore, a value of x_e indicates the important level of e to the network desired functionality.

Related Work and Challenges. Since `cLPI` is a new problem, it does not have much related work. Indeed, solving `cLPI` with bounded performance guarantee is challenging. First, a simple solution, which discretizes functions f_e and directly adopts the solutions of `QoSD`, actually has a problem. The discretization of f_e is simply a work of taking an integer x and returning the value $f_e(x \cdot \delta)$, where δ is called discretizing step. If δ is too large, the returned solution will be far from optimal due to discretization error; otherwise small δ creates significantly large inputs for `QoSD`, causing a burden on memory usage and undesirable running time. Therefore, a solution, which can directly applied into continuous domain, is more desired. Second, f_es could be any function, thus a typical Convex Optimization [4, 19] solution cannot be applied. Also, any solution for Constrained Optimization can easily fall into local optima with complicated f_es, so no performance ratio is guaranteed. Furthermore, enumerating all constraints, each is corresponding to a path in the network, is intractable as the number of paths can be upto $\sum_{k=2}^{n} \binom{n}{k} k!$ where n is number of nodes in the network.

Our contributions. This paper introduces and investigates the `cLPI` problem. Accordingly, we propose a general framework for solving `cLPI`, separating tasks into two different oracles, called *Critical Paths Listing* (`CPL`) and *Threshold Blocking* (`TB`). `CPL`'s job is to restrict the amount of paths considered for finding feasible solution of `cLPI`. `TB` deals with the task of finding $\mathbf{x}$ in continuous domain, guaranteeing all paths, returned by `CPL`, have length exceed a certain threshold. We next propose Threshold Expansion for the `TB` oracle and Feasible Set Construction to for `CPL`. Finally, we show that our solution obtain an approximation ratio which allows a user to control the trade-off between running time versus accuracy.

2 Preliminaries

2.1 Problem Formulation

We abstract a network using a directed graph $G = (V, E)$ with $|V| = n$ nodes and $|E| = m$ directed edges. Each edge e is associated with a function $f_e : \mathbb{R}^{\geq} \rightarrow \mathbb{R}^{\geq}$ which indicates the weight of e w.r.t a budget distributed for e. In another word, if we spend x on edge e, the weight of edge e will become $f_e(x)$. f_e is monotonically increasing for all $e \in E$.

A budget distribution contains budget for each edge. Thus, given an arbitrary order of edges $E = \{e_1, ...e_m\}$, we denotes a budget distribution under the form of a vector $\mathbf{x} = \{x_1, ...x_m\}$ where x_i is a budget spent on the edge e_i. For simplicity, we use the notation e to present an edge in E and its index in E also. So x_e means the budget spent on edge e, and the entry in $\mathbf{x}$ corresponding to e also. The overall budget on all edges, therefore, is $\|\mathbf{x}\| = \sum_{e \in E} x_e$.

A path $p = \{u_0, u_1, ...u_l\} \in G$ is a sequence of vertices such that $(u_{i-1}, u_i) \in E$ for $i = 1, .., l$. A path can also be understood as the sequence of edges $\{(u_0, u_1), (u_1, u_2), ...(u_{l-1}, u_l)\}$. In this work, a path is used interchangeably as a sequence of edges or a sequence of nodes. A *single path* is a path that there exists no node who appears more than one in the path. Under a budget vector $\mathbf{x}$, the length of a path p is defined as $\sum_{e \in p} f_e(x_e)$. `cLPI` is formally defined as follows:

Definition 1. Length-bounded Paths Interdiction in Continuous Domain (`cLPI`). *Given a directed graph $G = (V, E)$, a set $f = \{f_e : \mathbb{R}^{\geq} \to \mathbb{R}^{\geq}\}$ of edge weight functions and a target set of pairs of nodes $S = \{(s_1, t_1), ...(s_k, t_k)\}$, determine a budget distribution $\mathbf{x}$ with minimum budget $\|\mathbf{x}\|$ such that under $\mathbf{x}$, any path connecting a pair of S has length at least T.*

For each pair $(s, t) \in S$, we call s a *start node* and t a *end node*. Let $\mathcal{P}_i$ denote a set of simple paths connecting the pair $(s_i, t_i) \in S$, whose initial length do not exceed T, i.e $\sum_{e \in p} f_e(0) < T$ for all $p \in \mathcal{P}_i$. Let $\mathcal{F} = \cup_{i=1}^{k} \mathcal{P}_i$, we call a path $p \in \mathcal{F}$ a *feasible path* and $\mathcal{F}$ is a set of all feasible paths in G. A non-feasible path either connects no pair in S or has initial length exceed T.

Before going further, we now look at several notations, mathematical operators on vector space $\mathbb{R}^m$, which are used along the theoretical proofs of our algorithms. Given $\mathbf{x} = \{x_1, ...x_m\}, \mathbf{y} = \{y_1, ...y_m\} \in \mathbb{R}^m$, we have:

$$\mathbf{x} + \mathbf{y} = \{x_1 + y_1, ...x_m + y_m\}$$
$$\mathbf{x} \setminus \mathbf{y} = \{\max(x_1 - y_1, 0), ... \max(x_n - y_n, 0)\}$$

Moreover, we say $\mathbf{x} \leq \mathbf{y}$ if $x_i \leq y_i$ for all $i \in [1, m]$, the similar rule is applied to $<, \geq, >$.

Node version of the problem. The node version of `cLPI` asks for the minimum budget to increase node weights (rather than edge weights) in the problem definition above. Our solution can be easily adapted to the node version and keep the same theoretical performance guarantee.

2.2 General model of our solutions

In this part, we present an overview model of our solutions, including a general framework and its performance guarantee.

About performance guarantees, given the problem instance with a threshold T, denote `OPT` as an optimal solution. We call a budget distribution $\mathbf{x}$ is **ε-feasible** to `cLPI` iff under $\mathbf{x}$, the distance between each target pair is at least $T - \varepsilon$. Our algorithms are bicriteria approximation algorithms, returning a ε-feasible solution $\mathbf{x}$ whose overall budget is bounded within a factor $A(G, \varepsilon^{-1})$ of `OPT`, where $A(G, \varepsilon^{-1})$ depends on structure of the input graph and is monotone increasing with ε^{-1}. ε is treated as a trade-off between the algorithms' accuracy and running time. To be specific, the smaller ε is, the closer pairs' distances are to T but the longer it takes for the algorithms to finish. ε is adjustable, allowing users to control running time versus accuracy as desired.

About general framework, our solutions contain two separate oracles, called *Threshold Blocking* (`TB`) and *Critical Paths Listing* (`CPL`). These two oracles communicate back and forth with the other to construct a solution to `cLPI`, given an input instance of `cLPI` and a parameter ε. These two oracles are proposed to tackle two challenges of `cLPI` as stated before, to be specific:

– *Threshold Blocking* - a primary role of `TB` is to solve a sub-problem of `cLPI`: Given a target set $\mathcal{P}$ of single paths and a threshold $T_u \leq T$, `TB` aims to find a minimum additional budget on edges in order to make each path in $\mathcal{P}$ has length exceeding T_u. For simplicity, we call this task `TB` problem.

– *Critical Paths Listing* - this oracle restricts the number of paths which need to be considered in the algorithm, thus significantly reducing the searching space and burdens on algorithms' running time and memory for storage.

Separating into two oracles allows us to design different solutions to each of the oracles. Assume if there exists one solution for each oracle, the flow of our solution is as follows:

1. The algorithm starts with $x_e = 0$ for all $e \in E$ (i.e. $\mathbf{x} = \{0\}_e$).
2. Given the current state of $\mathbf{x}$, by using a technique to restrict searching space, `CPL` oracle searches for a set of critical paths, who are feasible paths and shorter than a pre-determined threshold $T_u \leq T$.
3. Then those paths along with a current state of $\mathbf{x}$ are given as an input for the `TB` oracle, which then finds an additional budget $\mathbf{v}$ for $\mathbf{x}$ to make all input paths' length exceed T_u.
4. The additional budget $\mathbf{v}$ is then used for `CPL` to check the feasibility. If adding $\mathbf{v}$ makes $\mathbf{x}$ ε-feasible, the algorithm returns $\mathbf{x}+\mathbf{v}$ and terminates. Otherwise, $\mathbf{v}$ is used to drive the searching space of `CPL` and find a new value for $\mathbf{x}$ and T_u; then step (2) is repeated.

Due to the space limit, we only present one solution to each oracle.

3 Threshold Blocking Oracle

In this section, we present our solution to the Threshold Blocking (`TB`) Oracle, called *Threshold Expansion* (`TE`).

3.1 Requirements of `TB`

To recap, `TB` receives a set $\mathcal{P}$ of critical paths from `CPL`, the current budget $\mathbf{x}$ and an upper threshold T_u. The objective of `TB` is to find an additional budget vector $\mathbf{v} = \{v_1, ...v_m\}$ with minimum $\sum_e v_e$ such that under the budget $\mathbf{x}+\mathbf{v} = \{x_e+v_e\}_e$, each path in $\mathcal{P}$ has length exceeding T_u, i.e. $\sum_{e \in p} f_e(x_e+v_e) \geq T_u$ for all $p \in \mathcal{P}$. Another information that `TB` gets is $T_l \geq 0$, which is a lower bound of each path's length, i.e $\sum_{e \in p} f_e(x_e) \geq T_l$ for all $p \in \mathcal{P}$. Without lost of generality, we assume that each path in $\mathcal{P}$, under $\mathbf{x}$, has length in range $[T_l, T_u)$.

The bicriteria guarantee of our algorithms originates from the `TB` algorithms. The desired accuracy ε is given to the `TB` oracle so the `TB` algorithm guarantees each path in $\mathcal{P}$ has length at least $T_u - \varepsilon$. To do so, an objective function of `TB` is defined as follows:

$$b_{\mathcal{P},\mathbf{x}}(\mathbf{v}) = \sum_{p \in \mathcal{P}} \min\Big(\sum_{e \in p} f_e(x_e + v_e), T_u\Big)$$

Trivially, a budget vector $\mathbf{v}$ satisfies `TB`'s objective iff $b_{\mathcal{P},\mathbf{x}}(\mathbf{v}) = |\mathcal{P}| \times T_u$. $b_{\mathcal{P},\mathbf{x}}(\cdot)$ can be seen as a function with $m = |E|$ variables. Let's take more insight into $b_{\mathcal{P},\mathbf{x}}(\cdot)$ as it is important for devising algorithms in the `TB` oracle. Define:

$$l_{p,\mathbf{x},e}(x) = \sum_{e' \in p \& e' \neq e} f_{e'}(x_{e'}) + \mathbf{1}_{e \in p} f_e(x_e + x)$$

$$r_{\mathcal{P},\mathbf{x},e}(x) = \sum_{p \in \mathcal{P}} \left(\min\Big(T_u, l_{p,\mathbf{x},e}(x)\Big) - \min\Big(T_u, l_{p,\mathbf{x},e}(0)\Big) \right)$$

Basically, $r_{\mathcal{P},\mathbf{x},e}(\cdot)$ measures the increasing value of $b_{\mathcal{P},\mathbf{x}}(\{0\})$ by adding a budget of x into entry e. It is easy to see that $r_{\mathcal{P},\mathbf{x},e}(x)$ is a monotone increasing function w.r.t x.

Assuming $\{p_1, ...p_l\} \subseteq \mathcal{P}$ are paths containing e and are sorted in descending order w.r.t to their length under $\mathbf{x}$. Define a_i as a minimum additional budget on edge e to make path p_i's length exceed T_u, i.e. $a_i = \arg\min_x \Big\{ l_{p_i,\mathbf{x},e}(x) \geq T_u \Big\}$. Let $a_0 = 0$. $\{a_i\}$ are in ascending order. $r_{\mathcal{P},\mathbf{x},e}(x)$ can be rewritten as:

$$r_{\mathcal{P},\mathbf{x},e}(x) = i \cdot T_u + \sum_{j>i} l_{p_j,\mathbf{x},e}(x) - Q_{\mathcal{P},\mathbf{x},e} \text{ with } a_i \leq x \leq a_{i+1} \qquad (1)$$

where $Q_{\mathcal{P},\mathbf{x},e} = \sum_{p;e \in p} \sum_{e' \in p} f_{e'}(x_{e'})$, which does not depend on either x or i. Equ. (1) allows us to discard the min term in the original $r_{\mathcal{P},\mathbf{x},e}(\cdot)$ to exploit the function's property within each range $[a_i, a_{i+1}]$.

3.2 Threshold Expansion

In a nutshell, our Threshold Expansion algorithm, `TE`, is a threshold greedy algorithm which aims to tackle the continuous domain challenges, especially when the objective function is not concave. `TE` starts with setting a sufficient large value of M, which is the upper bound of $\frac{r_{\mathcal{P},\mathbf{w},e}(x)}{x}$ for all $e \in E$, $x \geq 0$ and $\mathbf{w} \geq \mathbf{x}$. To find M, the algorithm utilizes the fact that $f_e(\cdot)$ is continuous and differentiable everywhere for all $e \in E$ as the following lemma.

Lemma 1. *By setting $M = |\mathcal{P}| \times \max_{x \geq 0, e \in E, f_e(x) \leq T_u} \frac{\partial f_e}{\partial x}$, the `TE` algorithm guarantees*

$$M \geq \frac{r_{\mathcal{P},\mathbf{w},e}(x)}{x} \textit{ for all } \mathbf{w} \geq \mathbf{x}, e \in E, x \geq 0$$

We omit this proof due to space limit.

The algorithm works on top of the $\mathbf{x}'$ vector, which is just a copy of $\mathbf{x}$ initially. This step is to separate the work on $\mathbf{x}$ between the `TB` and `CPL` oracle, e.g. `CPL` may not accept the result of `TB` (which is shown in the `CPL` section). Edges in E are sorted in an arbitrary order. `TE` considers edges sequentially in that order and for each edge e, `TE` finds a maximum addition budget $\hat{x}$ for e such that $\frac{r_{\mathcal{P},\mathbf{x}',e}(\hat{x})}{\hat{x}} \geq M$ and add $\hat{x}$ into e.

Different to previous work in the discrete domain and submodular maximization, the function $\frac{r_{\mathcal{P},\mathbf{x}',e}(\hat{x})}{\hat{x}}$ is not monotone increasing. Thus the technique of using binary search as in [21] is no longer applicable. To find $\hat{x}$, we utilize Equ. (1) by identifying local extreme points of $\frac{r_{\mathcal{P},\mathbf{x}',e}(x)}{x}$ within each range $[a_i, a_{i+1}]$ using

Algorithm 1 Threshold Expansion

Input

- $G = (V, E)$ - the input graph
- $f_e : \mathbb{R}^{\geq} \rightarrow \mathbb{R}^{\geq}$ for all $e \in E$
- $\mathcal{P}$ - the set of paths
- T_u - target threshold
- ϵ - the performance parameter of `TE`
- ε - the accuracy parameter
- $\mathbf{x}$ - current budget vector

Output: $\mathbf{v}$ - an additional budget vector to $\mathbf{x}$ to make each path in $\mathcal{P}$ has length exceeding $T_u - \varepsilon$

```
1: Sort $E$ in an arbitrary order
2: $\mathbf{v} = \{0\}$, $\mathbf{x}' = \mathbf{x}$
3: $M = |\mathcal{P}| \times \max_{x \geq 0, e \in E, f_e(x) \leq T_u} \frac{\partial f_e}{\partial x}$
4: $e \leftarrow$ the first edge in $E$
5: while $\exists p \in \mathcal{P}$ that $p$'s length $< T_u - \varepsilon$ do
6:     $\hat{x} = \arg\max_x \left\{ \frac{r_{\mathcal{P},\mathbf{x}',e}(x)}{x} \geq M \right\}$
7:     $\mathbf{l} =$ a vector with $\hat{x}$ at entry $e$ and 0 elsewhere
8:     $\mathbf{x}' = \mathbf{x}' + \mathbf{l}$, $\mathbf{v} = \mathbf{v} + \mathbf{l}$
9:     if $e$ is the last edge in $E$ then
10:        $M = (1 - \epsilon)M$
11:        $e \leftarrow$ start over with the first edge
12:    else
13:        $e \leftarrow$ the next edge.
```

Return $\mathbf{v}$

the function's first derivative and exploiting the increasing/decreasing traits of the function. Note that there could be a case that $\hat{x}$ cannot be found, if so we set $\hat{x} = 0$ and no budget is added into the considered edge. After adding $\hat{x}$ into e, the algorithm considers the next edge.

After the algorithm has considered the last edge in E in the order as stated, it means the algorithm has finished a round of edges, `TE` reduces M by a factor of $1 - \epsilon$ and starts over with the first edge in the order. Whenever `TE` adds a budget into an edge, the algorithm constantly checks whether $\mathbf{v}$ is sufficient to make each path's length exceed $T_u - \varepsilon$ and terminates whenever this condition is satisfied. The pseudo-code of `TE` is presented in Alg. 1.

The adaptation into the continuous domain of `TE` can be seen as in the way the algorithm works. We now turn our attention to `TE`'s performance guarantee. From now on, for simplicity, when we analyze the performance of the algorithm at a certain moment when it is running, we refer M, $\mathbf{v}$ and $\mathbf{x}'$ as their values at that moment.

Let's consider at a certain moment, denote $\mathbf{v}^o = \{v_e^o\} = \mathbf{v}^* \setminus \mathbf{v}$. We have the following lemma.

Lemma 2. *$v_e^o = 0$ or $\frac{r_{\mathcal{P},\mathbf{x}',e}(v_e^o)}{v_e^o} < \frac{M}{1-\epsilon}$ for all $e \in E$.*

We omit this proof due to space limit.

Therefore, even the edge weight functions are not concave or $\frac{r_{\mathcal{P},\mathbf{x'},e}(\hat{x})}{\hat{x}}$ is not monotone increasing, the selection of $\hat{x}$ and Lemma 2 allow us to bound the performance guarantee of `TE`, which is shown in the following theorem.

Theorem 1. *Given the information $G, f_e, \mathcal{P}, T_u, T_l, \varepsilon, \mathbf{x}$, if $\mathbf{v}$ is the budget returned by* `TE` *and $\mathbf{v}^*$ is the minimum additional budget to make each path in $\mathcal{P}$ has length exceeding T_u, then:*

$$\|\mathbf{v}\| \leq \frac{\ln\left(|\mathcal{P}|(T_u - T_l)\varepsilon^{-1}\right) + 1}{1-\epsilon}\|\mathbf{v}^*\|$$

Proof. Let's assume edge e is being considered and $\hat{x}$ is the selected amount to add into e. Again, denote $\mathbf{v}^o = \{v^o_e\} = \mathbf{v}^* \setminus \mathbf{v}$. Without lost of generality, let $\hat{x} > 0$. From lemma. 2, we have:

$$\frac{r_{\mathcal{P},\mathbf{x'},e}(\hat{x})}{\hat{x}} \geq (1-\epsilon)\frac{r_{\mathcal{P},\mathbf{x'},e}(v^o_{e'})}{v^o_{e'}}$$

for all $e' \in E$ that $v^o_{e'} > 0$

Denote $\mathbf{x'} = \{x_e\}_{e\in E}$, $\mathbf{h}_e = \{x_{e'} + \mathbf{1}_{e'>e}v^o_{e'}\}_{e'\in E}$. As $\mathbf{h}_e \geq \mathbf{x'}$ but they have the same value at entry e, we have:

$$r_{\mathcal{P},\mathbf{h}_e,e}(v^o_e) \leq r_{\mathcal{P},\mathbf{x'},e}(v^o_e)$$

Therefore,

$$b_{\mathcal{P},\mathbf{x'}}(\mathbf{v}^o) - b_{\mathcal{P},\mathbf{x'}}(\{0\}) = \sum_{e\in E} r_{\mathcal{P},\mathbf{h}_e,e}(v^o_e) \leq \sum_{e\in E} r_{\mathcal{P},\mathbf{x'},e}(v^o_e)$$

$$\leq \sum_{e'\in E}\frac{v^o_{e'}}{\hat{x}(1-\epsilon)}r_{\mathcal{P},\mathbf{x'},e}(\hat{x}) \leq \frac{\|\mathbf{v}^*\|}{\hat{x}(1-\epsilon)}r_{\mathcal{P},\mathbf{x'},e}(\hat{x})$$

Note that $b_{\mathcal{P},\mathbf{x'}}(\mathbf{v}^o) = |\mathcal{P}| \times T_u$.

Now, let's assume the algorithm terminates after adding budget into edges L times, denote $\hat{x}_1, ...\hat{x}_L$ as an added budget at each times ($\|\mathbf{v}\| = \sum_{i=1}^{L}\hat{x}_i$). Also, denote $\mathbf{x}'_t$, $\mathbf{v}_t$ as $\mathbf{x}'$, $\mathbf{v}$ before adding $\hat{x}_t$ at time t. We have:

$$|\mathcal{P}| \times T_u - b_{\mathcal{P},\mathbf{x}}(\mathbf{v}_t) = |\mathcal{P}| \times T_u - b_{\mathcal{P},\mathbf{x}'_t}(\{0\}) \leq \frac{\|\mathbf{v}^*\|}{\hat{x}_t(1-\epsilon)}r_{\mathcal{P},\mathbf{x}'_t,e}(\hat{x}_t)$$

$$= \frac{\|\mathbf{v}^*\|}{\hat{x}_t(1-\epsilon)}\Big(b_{\mathcal{P},\mathbf{x}}(\mathbf{v}_{t+1}) - b_{\mathcal{P},\mathbf{x}}(\mathbf{v}_t)\Big)$$

Thus:

$$|\mathcal{P}| \times T_u - b_{\mathcal{P},\mathbf{x}}(\mathbf{v}_{t+1}) \leq \left(1 - \frac{\hat{x}_t(1-\epsilon)}{\|\mathbf{v}^*\|}\right)\Big(|\mathcal{P}| \times T_u - b_{\mathcal{P},\mathbf{x}}(\mathbf{v}_t)\Big)$$

Therefore, we have:

$$|\mathcal{P}| \times T_u - b_{\mathcal{P},\mathbf{x}}(\mathbf{v}_{L-1}) \leq \prod_{t=1}^{L-1}\left(1 - \frac{\hat{x}_t(1-\epsilon)}{\|\mathbf{v}^*\|}\right)\Big(|\mathcal{P}| \times T_u - b_{\mathcal{P},\mathbf{x}}(0)\Big)$$

$$\leq \left(1 - \frac{\sum_t^{L-1}\hat{x}_t(1-\epsilon)}{\|\mathbf{v}^*\|(L-1)}\right)^{L-1}|\mathcal{P}|(T_u - T_l) \leq e^{-\frac{\|\mathbf{v}^{L-1}\|}{\|\mathbf{v}^*\|}(1-\epsilon)}|\mathcal{P}|(T_u - T_l)$$

After $L-1$ updates, there should exist at least a path in $\mathcal{P}$ whose length is shorter than $T_u - \varepsilon$ (otherwise the algorithm should terminate after $L-1$ updates). Thus $|\mathcal{P}|T_u - b_{\mathcal{P},\mathbf{x}}(\mathbf{v}_{L-1}) \geq \varepsilon$, which means:

$$\|\mathbf{v}_{L-1}\| \leq \|\mathbf{v}^*\| \frac{\ln\Big(|\mathcal{P}|(T_u - T_l)\varepsilon^{-1}\Big)}{1-\epsilon}$$

Now, let consider the final update, we have:

$$\hat{x}_L \leq \frac{\|\mathbf{v}^*\|}{1-\epsilon} \cdot \frac{b_{\mathcal{P},\mathbf{x}}(\mathbf{v}_L) - b_{\mathcal{P},\mathbf{x}}(\mathbf{v}_{L-1})}{|\mathcal{P}| \times T_u - b_{\mathcal{P},\mathbf{x}}(\mathbf{v}_{L-1})} \leq \frac{\|\mathbf{v}^*\|}{1-\epsilon}$$

Finally, we have:

$$\|\mathbf{v}\| = \|\mathbf{v}_{L-1}\| + \hat{x}_L \leq \|\mathbf{v}^*\| \frac{\ln\Big(|\mathcal{P}|(T_u - T_l)\varepsilon^{-1}\Big) + 1}{1-\epsilon}$$

which completes the proof.

4 Critical Path Listing Oracle

Algorithm 2 Feasible Set Construction

Input

- $G = (V, E)$ - the input graph
- $f_e : \mathbb{R}^{\geq} \to \mathbb{R}^{\geq}$ for all $e \in E$
- T - target threshold
- ε - accuracy parameter
- S - set of target pairs
- `TB` - threshold blocking oracle

Output x

```
1:  $\mathcal{P} = \emptyset, \mathbf{x} = \mathbf{v} = \{0\}$
2:  while $\exists(s,t) \in S$ that $d(s,t) < T - \varepsilon$ in $G$ do
3:      Construct shortest path trees for all start nodes
4:      $\mathcal{L} = \emptyset$
5:      for each pair $(s,t) \in S$ do
6:          $H \leftarrow$ a copy of shortest path tree with root $s$
7:          $X = \emptyset$
8:          while $d(s,t) < T - \varepsilon$ in $H$ do
9:              $p \leftarrow$ the shortest path from $s$ to $t$ in $H$
10:             $\mathcal{L} = \mathcal{L} \cup \{p\}$
11:             Randomly pick $e = (u,v) \in p$ and put into $X$
12:             Reconstruct $H$ without edges of $X$
13:     $\mathcal{P} = \mathcal{P} \cup \mathcal{L}$
14:     $\mathbf{v}$ = run TB oracle with input $G, f_e, \mathcal{P}, T, \varepsilon, \mathbf{x}$
15:     Set edge $e$'s weight to be $f_e(v_e)$ for all $e \in E$
```

Return v

In this section, we present *Feasible Set Construction* (`FC`) for the `CPL` oracle. The role of the `CPL` oracle is to reduce the searching space when constructing the returned solution $\mathbf{x}$. It works as a backbone for the overall process of finding $\mathbf{x}$. It is the one receiving the input information of the cLPI problem, then communicating back and forth with TB to construct $\mathbf{x}$ and return $\mathbf{x}$ when $\mathbf{x}$ guarantees that a distance between each target pair exceeds $T - \varepsilon$.

In general, FC (shown in Alg. 2) aims to construct a set $\mathcal{P}$ of candidate paths, which is a subset of $\mathcal{F}$ but, if being used as an input for TB with a threshold $T_u = T$, can return $\mathbf{v}$ that is a ε-feasible solution of cLPI. $\mathcal{P}$ is constructed in order to avoid fully listing all paths in $\mathcal{F}$ when $\mathcal{F}$ is significantly large. FC starts with $\mathcal{P} = \emptyset$ and then builds it incrementally and iteratively. For each iteration, the algorithm uses the TB oracle to find a budget vector $\mathbf{x}$ to make each path in $\mathcal{P}$ has length exceeding $T - \varepsilon$. Then, the length of an edge e is set to be $f_e(x_e)$. Next, FC checks whether $\mathbf{x}$ is ε-feasible. If not, the algorithm adds a set $\mathcal{L}$ of feasible paths into $\mathcal{P}$ where $\mathcal{L}$ contains paths, each of whom has length shorter than $T - \varepsilon$ ($\mathcal{L} \cap \mathcal{P} = \emptyset$); then reset all edges' length (i.e. the length of e turns back to $f_e(0)$). If yes, the algorithm returns $\mathbf{x}$ and terminates.

To optimize the number of iterations, after updating e's length to be $f_e(x_e)$ for all $e \in E$, FC builds shortest-path trees, a root of each tree is a *start node* of a target pair. Each pair $(s,t) \in S$ is then associated with a copy H of the shortest-path tree rooted at s and a set X of edges, which is initially set to be empty. For each pair $(s,t) \in S$, the algorithm works in an iterative fashion:

1. FC adds the shortest path $p \subseteq H$ from s to t into $\mathcal{L}$.
2. Then FC randomly picks an edge $e = (u,v) \in p$ and put e into X.
3. A sub-tree of H rooted at u is re-constructed with the condition that H contains no edge in X.
4. If there still exists a path from s to t in H with length shorter than $T - \epsilon$, the algorithm is back to step (1).

We have the following theorem.

Theorem 2. *The approximation guarantee of FC equals to the approximation guarantee of the algorithm used in the TB oracle.*

Proof. This Theorem uses a similar concept as Lemma 4.1 [20], in which we observe that: Since $\mathcal{P}$ is a subset of $\mathcal{F}$, the optimal budget $\|\mathbf{x}^*\|$ to cLPI is at least the optimal budget $\|\mathbf{x}^o\|$ to make all paths' length of $\mathcal{P}$ exceed T. Denote α as an approximation guarantee of the TB oracle, i.e. $\|\mathbf{x}\| \leq \alpha\|\mathbf{x}^o\|$. As $\mathbf{x}$ is guaranteed to be ε-feasible to the cLPI instance, α is also the approximation guarantee of FC to cLPI.

Combining Theorems 1 and 2, our solution to cLPI has a bicriteria approximation ratio of $\big(\ln(|\mathcal{P}_{\mathtt{TE}}| \cdot T \cdot \varepsilon^{-1}) + 1\big)(1-\epsilon)^{-1}$, where $\mathcal{P}_{\mathtt{TE}}$ is the final sets $\mathcal{P}$ of candidate paths using TE.

5 Conclusion

In this paper, we introduced the cLPI problem, which allows us to better assess the modern network vulnerability. To tackle the challenges of cLPI, we developed a solution framework that consists of two oracles, namely *Threshold Blocking* (`TB`) oracle and *Critical Path Listing* (`CPL`) oracle, which communicate back and forth to construct a feasible solution with theoretical performance guarantees. We further devised a bicriteria approximation algorithm to `cLPI`, of which we offer one solution to each oracle. For future work, we may consider different variants of `cLPI`. For example, an edge could be associated with multiple functions, serving for multiple objectives of networked functionality. Also, each function can have multiple variables and each variable could appear on more than one functions, making the problem become much more complicated. A solution, which can balance multiple objectives, is desirable. Furthermore, another perspective considering network flows is of interest, which we aim to modify edge weights to guarantee the max flow of the network is at most a certain threshold.

Acknowledgements

This work was supported in part by NSF CNS-1814614 and NSF IIS-1908594.

References

1. Priority packet. `https://www.sciencedirect.com/topics/computer-science/priority-packet` (2019), accessed: 2019-07-18
2. Andreev, K., Racke, H.: Balanced graph partitioning. Theory of Computing Systems **39**(6), 929–939 (2006)
3. Birge, J.R., Louveaux, F.V.: A multicut algorithm for two-stage stochastic linear programs. European Journal of Operational Research **34**(3), 384–392 (1988)
4. Boyd, S., Vandenberghe, L.: Convex optimization. Cambridge university press (2004)
5. Chawla, S., Krauthgamer, R., Kumar, R., Rabani, Y., Sivakumar, D.: On the hardness of approximating multicut and sparsest-cut. computational complexity **15**(2), 94–114 (2006)
6. Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., Kohno, T., et al.: Comprehensive experimental analyses of automotive attack surfaces. In: USENIX Security Symposium. pp. 77–92. San Francisco (2011)
7. Chen, Q.A., Yin, Y., Feng, Y., Mao, Z.M., Liu, H.X.: Exposing congestion attack on emerging connected vehicle based traffic signal control. In: Network and Distributed Systems Security (NDSS) Symposium 2018 (2018)
8. Dinh, T.N., Thai, M.T.: Precise structural vulnerability assessment via mathematical programming. In: 2011-MILCOM 2011 military communications conference. pp. 1351–1356. IEEE (2011)
9. Dinh, T.N., Thai, M.T.: Assessing attack vulnerability in networks with uncertainty. In: Computer Communications (INFOCOM), 2015 IEEE Conference on. pp. 2380–2388. IEEE (2015)
10. Dinh, T.N., Thai, M.T.: Network under joint node and link attacks: Vulnerability assessment methods and analysis. IEEE/ACM Transactions on Networking **23**(3), 1001–1011 (2015)

11. Fanti, G., Viswanath, P.: Deanonymization in the bitcoin p2p network. In: Advances in Neural Information Processing Systems. pp. 1364–1373 (2017)
12. Garg, N., Vazirani, V.V., Yannakakis, M.: Approximate max-flow min-(multi) cut theorems and their applications. SIAM Journal on Computing **25**(2), 235–251 (1996)
13. Garg, N., Vazirani, V.V., Yannakakis, M.: Primal-dual approximation algorithms for integral flow and multicut in trees. Algorithmica **18**(1), 3–20 (1997)
14. Jeske, D.R., Sampath, A.: Signal-to-interference-plus-noise ratio estimation for wireless communication systems: Methods and analysis. Naval Research Logistics (NRL) **51**(5), 720–740 (2004)
15. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C.: Optimization by simulated annealing: an experimental evaluation; part i, graph partitioning. Operations research **37**(6), 865–892 (1989)
16. Keiser, G.: Optical fiber communications. Wiley Encyclopedia of Telecommunications (2003)
17. Kuhnle, A., Crawford, V.G., Thai, M.T.: Network resilience and the length-bounded multicut problem: Reaching the dynamic billion-scale with guarantees. Proceedings of the ACM on Measurement and Analysis of Computing Systems **2**(1), 4 (2018)
18. Lee, E.: Improved hardness for cut, interdiction, and firefighter problems. arXiv preprint arXiv:1607.05133 (2016)
19. Nesterov, Y.: Lectures on convex optimization, vol. 137. Springer (2018)
20. Nguyen, L.N., Thai, M.T.: Network resilience assessment via qos degradation metrics: An algorithmic approach. Proceedings of the ACM on Measurement and Analysis of Computing Systems **3**(1), 1 (2019)
21. Soma, T., Yoshida, Y.: Maximizing monotone submodular functions over the integer lattice. Mathematical Programming **172**(1-2), 539–563 (2018)
22. Svitkina, Z., Tardos, É.: Min-max multiway cut. In: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, pp. 207–218. Springer (2004)