Thesis for the degree
Master of Science

Submitted to the Scientific Council of the
Weizmann Institute of Science
Rehovot, Israel

By
Itay Gonshorovitz

עבודת גמר (תזה) לתואר
מוסמך למדעים

מוגשת למועצה המדעית של
מכון ויצמן למדע
רחובות, ישראל

מאת
איתי גונשורוביץ

הקטנת הזרימה המקסימלית
Reducing The Maximum Flow

Advisor:
Prof. Uriel Feige

מנחה:
פרופ' אוריאל פייגה

November 2012

כסלו תשע"ג

Abstract

We study two problems in which the goal is to reduce the maximum s-t flow in a flow network, while paying the minimum cost. In the first problem, only one edge can be removed, and the goal is to remove an edge with the minimum cost whose removal decreases the maximum s-t flow. This problem is in **P**, and we give an algorithm with a running time of $O(|V|)$ max flow computations to solve it. The second problem is more general. Any subset of edges may be removed and the goal is to find the subset with the minimum total cost whose removal decreases the maximum s-t flow to at most a given threshold k. This problem is NP-hard, and we give two approximation algorithms for it. The first algorithm is a greedy $|E|$-approximation. The second algorithm is a bi-criteria approximation. Given $\epsilon > 0$, it outputs either a solution with a cost of at most $(1+\epsilon)$ times that of the optimal solution that decreases the maximum flow to at most k, or a solution with a cost of at most that of the optimal solution that decreases the maximum flow to at most $(1 + \frac{1}{\epsilon})k$.

Acknowledgments

I would like to thank my advisor, Uriel Feige, for his patient guidance and encouragement, for making things clear and simple, and for being a great teacher.

I would also like to thank the faculty and the institute for providing a peaceful studying environment.

The last two years have been very pleasant thanks to my cubic friends: Roee, Eldad and Motke. I enjoyed discussing with Itai Benjamini, and working with Tsvi Kopolowitz.

Special thanks to Hagar.

This work was supported by the Citi Foundation.

1 Introduction

We study two problems in which the goal is to reduce the maximum flow in a flow network between two nodes. Removing an edge comes with a cost, and we need to pick edges with the minimum total removal cost. The first problem we study is *Min Necessary Edge* in which we are allowed to remove only one edge. Then we analyze more general problem which we call the *Flow Reduction Problem*. In this problem any subset of edges can be removed and the goal is to find the subset of edges with the minimum cost such that removing this subset of edges decreases the flow to at most some threshold. Below are the formal definitions of theses two problems:

Definition 1 ***(Min Necessary Edge Problem).*** *The input for the min necessary edge problem is a directed graph $G = (V, E)$ with a capacity c_e and a removal cost r_e associated with each edge $e \in E$, a source node s and a target node t with a maximum flow f between them. The goal is to find an edge $e \in E$ with a minimum cost such that removing e from G reduces the max flow from s to t below $|f|$.*

Definition 2 ***(Flow Reduction Problem).*** *The input for the flow reduction problem (denoted by st-k-FR) is a directed graph $G = (V, E)$ with a capacity c_e and a removal cost r_e associated with each edge $e \in E$, a source node s, a target node t and a threshold k. The goal is to find a subset of edges $E' \subseteq E$ with the minimum removal cost such that removing E' from E reduces the maximum flow between s and t to at most k. The cost of removing E' is defined as the sum of removal costs of its edges, $\sum_{e \in E'} r_e$.*

In both problems we assume that the capacities and removal costs of the edges, and the flow threshold are integral.

One motivation for this work comes from an Anti Money Laundering (AML) research initiative. The abstract setting that we have in mind involves three entities: a money banking system, a money launderer, and a law enforcement agency. Activity in the banking system is represented by a directed *Transaction Graph*. Vertices in the graph represent accounts, directed edges represent money transfers between the respective accounts, and every edge has a capacity indicating the amount of money transferred. (Additionally, edges are labeled by a time stamp indicating when the transaction was made. This aspect is important and handled within our framework, but details of this

are omitted in this overview.) Some of the money circulating in the banking system might come from illegal sources, and is referred to as *black* money. A money launderer might attempt to inject black money from his own accounts into the system, and have it intermingle with the legitimate money that is circulated. The law enforcement agency can perform *audits* on edges. Such an audit can detect (using means available to the law enforcement agency) whether the underlying transaction included black money. However, certain bank accounts effectively serve as money laundering facilities: any money that is transferred out of them appears to be legitimate, regardless of its original source. Namely, audits of transactions that involve black money that has already passed through a money laundering facility will not detect that it was black.

Suppose that the law enforcement agency suspects that account s injects more than k units of black money into the banking system (intermingled with legal money that s injects), and that account t is used in order to launder it. To substantiate this suspicion it needs to audit at least one edge (that we refer to as a *black* edge) in which some of this black money was transferred on its route from s to t (which might be an indirect route involving multiple intermediate accounts). The administrator of the banking system is ordered to disclose to the law enforcement agency a set E' of edges for auditing, with the guarantee that if indeed s transferred more than k units of black money to t, then E' surely includes at least one black edge. Disclosing an edge has some costs involved (e.g., administrative overhead, the cost in privacy of the respective clients of the bank), and hence the administrator would like to choose a set of edges of minimum cost. This is equivalent to choosing the least cost set of edges whose removal reduces the flow from s to t to at most k. This is precisely the flow reduction problem.

2 Results

It is easy to see that the min necessary edge problem can be solved in polynomial time. A trivial algorithm for this problem removes every edge and checks if the maximum flow decreases, and hence has a worst case running time of $|E|$ max flow computations. We provide an algorithm $\mathcal{A}$ with a better worst case running time.

Theorem 3 *Algorithm $\mathcal{A}$ finds solves min necessary edge problem in time of $O(|V|)$ max flow computations.*

As opposed to the min necessary edge problem, the flow reduction problem is NP-hard. We will review two reductions that demonstrate this: one from the knapsack problem and the other from k-clique. As the problem is NP-hard, our focus is on designing approximation algorithms for it. We provide two approximation algorithms. The first is a simple greedy algorithm.

Proposition 4 *There is a greedy algorithm that is an $|E|$-approximation for the flow reduction problem.*

The second algorithm is a bi-criteria approximation algorithm. The standard notion of an (α, β)-approximation (for $\alpha, \beta > 1$) for the the flow reduction problem asks for a solution whose removal cost is at most α times that of the optimal, while decreasing the maximum s-t flow to at most βk. Our approximation will have a stronger guarantee than a standard (α, β)-approximation. We denote this type of approximation by $(\alpha \vee \beta)$-approximation.

Definition 5 *An $(\alpha \vee \beta)$-approximation algorithm for the Flow Reduction Problem outputs one of the two solution:*

1. *A feasible solution (decreasing the maximum s-t flow to at most k) with a removal cost of at most α times that of the optimal solution.*
2. *A solution with a removal cost at most that of the optimal solution, while it decreases the maximum s-t flow to at most βk.*

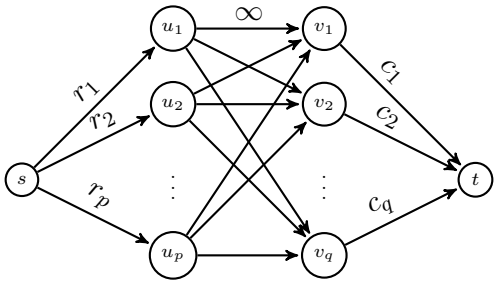
The second algorithm $\mathcal{B}$ is a $(\alpha \vee \beta)$-approximation, where there is a trade-off between the values of α and β.

Theorem 6 *For every $\epsilon > 0$, algorithm $\mathcal{B}$ provides a $((1+\epsilon) \vee (1+\frac{1}{\epsilon}))$-approximation for the flow reduction problem.*

3 Related Work

A related problem to the min necessary edge was studied in [12]. There the problem was to determine the minimum and the maximum flow of an edge in a maximum feasible

flow. The problem has arisen in the context of protecting private data while releasing statistics from this data. More specifically, given a matrix $A \in \mathbb{N}^{p \times q}$ with nonnegative entries, the question is what can be deduced about the value of each element a_{ij} by releasing the sum of each row r_i and the sum of each column c_j. For each a_{ij} we want to know what are the minimum and maximum values that are possible in order to be consistent with the disclosed data. The following network was used in order to find the possible minimum values for the entries.



Each edge (u_i, v_j) corresponds to a matrix element a_{ij} and its capacity is set to infinity. The maximum flow has a size of $\sum_i r_i = \sum_j c_j$ as such a flow gives values for the matrix entries which are consistent with the rows and columns sums. A necessary edge in our problem is equivalent to an edge with a minimum flow that is bigger than zero in that network. Hence, by finding the minimum flow value for an edge we can know if it is necessary or not. In the paper, the minimum values of the pq middle edges are found by $p + q + 1$ computations of maximum flow procedures. This result is similar to ours as $|V| = p + q + 2$ and pq is roughly $|E|$ for that graph. However, our problem deals with general graphs and also the fact the most of the edges in the above graph have infinite capacities makes the problem easier. Nevertheless, we will still use some similar techniques.

The flow reduction problem is strongly related to two problems. The first is the *Network Inhibition Problem* which was proposed independently by Wood [16] and by Phillips [14]. In this problem, we are given a network with capacities and costs for edges along with a budget B and the goal is to reduce the flow between two nodes as much as possible by removing edges with total cost of at most B. In our problem the goal is to minimize the budget which is a constraint for Network Inhibition Problem, while the constraint in our problem which is reducing the maximum flow to k is the goal for Network Inhibition problem. Hence, the two problem are the same up to renaming of

parameters. There are many flavors for the Network Inhibition problem. One of them, which is denoted by *NIP*, was studied by Burch et al. [4]. In that version, it is allowed to remove a fraction of capacity of an edge, while paying the fraction of its removal cost. They showed a $((1+\epsilon) \vee (1+\frac{1}{\epsilon}))$-approximation for it. An approximation for *st-k-FR* is equivalent to an approximation for a stricter version of *NIP*, in which edges are not allowed to be removed fractionally. Our approximation algorithm is inspired by the ideas of the approximation of *NIP*.

The second problem that is related to our problem is the k-route cut problem. The framework for multi-route flows and cuts was proposed by Kishimoto [13]. A k-multi-route flow is a flow that is robust against $k-1$ edge removal in the network. Similarly, a k-route cut for some terminals consists of removing a set of edges such that there are less than k disjoint paths between the terminals. In the *minimum multi-route cut problem* each edge has a removal cost and the goal is to find a k-route cut with the minimum cost. As in the standard cut problems, the k-route cut problems have many flavors such as st-cut, multicut etc..

The version of single source and single sink is called *(st)-EC-kRC* and it is a special case of our problem. In our problem the capacity for each edge is arbitrary while in the *(st)-EC-kRC* problem each edge has a unit capacity. Barman and Chawla [2] showed a $(4,4)$-bi-criteria approximation for the *(st)-EC-kRC* problem. Chuzhoy et al.[7] showed k-approximation and improved bi-criteria approximation of $(1+\frac{1}{c}, 1+c)$ for every $c>0$, which is essentially a $(1+\frac{1}{c} \vee 1+c)$ approximation. Given the optimal value of the optimal solution, the algorithm reduces the problem of approximating *(st)-EC-kRC* instance to solving a min-cut instance. In [4], there is a reference to unpublished manuscript by Rao, Shmoys and Tardos [5] for a parametric search algorithm for *NIP*. We describe a parametric search algorithm for *st-k-FR* in appendix A and compare it to our algorithm (the algorithm also solves min-cut problems to obtain the approximation). On the hardness side, it was also proved in [7] that under the Random k-AND assumption, it is NP-hard to approximate *(st)-EC-kRC* within any constant factor. This hardness result can be applied directly to our problem as well.

4 General Background

The problems we study are variants of the original maximum flow problem. The famous maximum flow problem was formulated by Ford and Fulkerson [8]. It is defined as:

Definition 7 *Let $G = (V, E)$ be a directed graph with a source node $s \in V$, and a target node $t \in V$. Each edge $e \in E$ has a nonnegative capacity c_e which represents the maximum flow that can pass through the edge. A flow f is a mapping $f : E \to R^+$, where f_e denotes the flow of edge $e \in E$, that consists of the following constraints:*

1. *Capacity constraints: for each edge $e \in E$ it holds that $0 \leq f_e \leq c_e$*
2. *Flow conservation: $\sum_{v:(u,v)\in E} f_{uv} - \sum_{v:(v,u)\in E} f_{vu} = 0$ for each $v \in V \setminus \{s, t\}$*

The value of the flow is $|f| = \sum_{u:(s,u)\in E} f_{su}$ and called the net flow. In the maximum flow problem the goal is to find a flow f such as $|f|$ is maximized.

The Ford-Fulkerson algorithm that was invented in 1955 was the first known algorithm to solve this problem and since then many other algorithms were developed. Most of the algorithms make use of the *Residual Graph* which is a network representing the remaining capacity for each edge after invoking a flow f.

Definition 8 *Given a network $G = (V, E)$ and a flow f the residual graph $G_f = (V, E_f)$ has the same vertices as in G. For each edge $(u, v) \in E$ with $f_{uv} < c_{uv}$ there is a residual edge $(u, v) \in E_f$ with $c^f_{uv} = c_{uv} - f_{uv}$, and for each edge $(u, v) \in E$ with $f_{uv} > 0$ there is a backward residual edge $(v, u) \in E_f$ with $c^f_{vu} = f_{uv}$.*

An augmenting path is a path from the source to sink in the residual graph. Note that if we have an augmenting path in G_f, then this means we can push additional flow along such a path in G beyond the flow f. Hence, if f' is a feasible flow in G_f then $f + f'$ is a feasible flow in G.

The min-cut problem is the dual of the max-flow problem.

Definition 9 *Let $G = (V, E)$ be a graph with a capacity c_e associated with each edge $e \in E$. An $s - t$ cut $C(S, T)$ is a partition of vertices into two sets S and T such that*

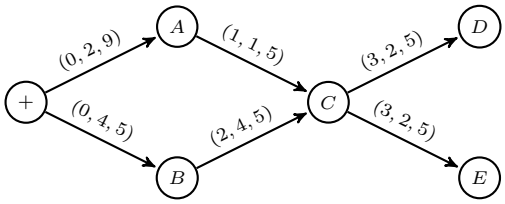
$s \in S$ and $t \in T$. The value of a cut $c(S,T)$ is $\sum_{\{(u,v):u\in S \wedge v\in T\}} c_{uv}$. In the min $s-t$ cut problem the goal is to find a partition $C(S,T)$ such that $c(S,T)$ is minimized.

Note that for directed graphs $c(S,T) \neq c(T,S)$ while for undirected graphs the cut value is a symmetric function. The min-cut max-flow theorem states that the value of the maximum flow from s to t is equal to the minimum cut between s and t.

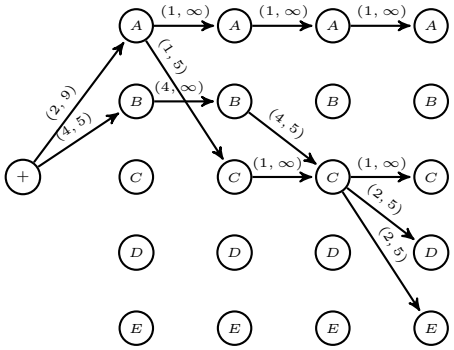
5 The Transaction Graph Model

The transaction graph is used to model the banking system in order to study questions about flow of money between bank accounts. Vertices in the graph correspond to bank accounts and edges correspond to transactions. Each edge contains information of its transactions which is the amount of money of the transaction, the time that the transaction has occurred and the cost to check the edge manually. In flow networks terminology, the amount of the money of a transaction can be viewed as the capacity for its corresponding edge as the flow between two accounts on a specific edge is bounded by the total amount of money for that transaction. In addition, the manual checking cost of a transaction can be viewed as a removal cost of an edge since if the transaction is found to be legal the edge can be removed from the graph as each flow of laundered money does not use that edge. We assume that a valid flow path of money has to be monotone increasing in time. In addition, we also assume that an account cannot have a negative balance at any time. To model it in the transaction graph, we add a special vertex which is denoted by "+". This vertex has an outgoing edge into all vertices with the amount of their initial balance at time zero. Below are the formal definition for the transaction graph and an example for such a graph.

Definition 10 *The transaction graph is a directed graph $G = (V, E)$. Each edge $e \in E$ has a capacity c_e, a removal cost r_e, and a timestamp t_e which are all nonnegative and integral. In addition, the transaction graph contains a special vertex, "+" $\in V$, which has only outgoing edges with timestamp of zero and infinite removal cost.*



For each edge the first value is the timestamp, the second is the capacity one and the third is the removal cost. In order to avoid complications that arise from the timestamps, the transaction graph can be transformed into a standard flow network. Let $G = (V, E)$ be a transaction graph, and let T be the maximum timestamp value for an edge in the graph. We transform G into a standard network $G' = (V', E')$ as follows. Each vertex $u \in V$ is duplicated into T copies $u_1 \ldots, u_T \in V'$ where $u_i \in V'$ represents the state of u at time i. An edge $(u, v) \in E$ with timestamp t, capacity c_{uv} and removal r_{uv} is transformed into an edge (u_t, v_{t+1}) in E' with the same capacity c_{uv} and removal cost r_{uv}. In addition, for each i such that $1 \leq i \leq T - 1$ and each $u \in V$ there is the edge (u_i, u_{i+1}) in G' if the balance of u in time i is positive. The capacity bound of that edge is equals to the positive value of the balance and it has an infinite removal cost.



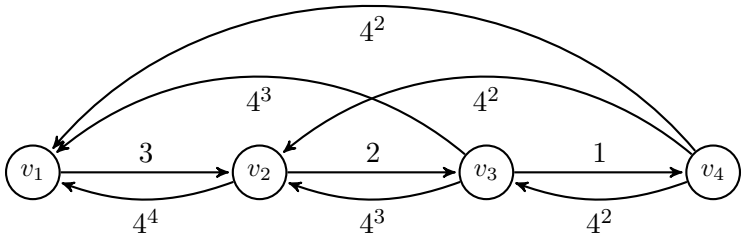
It is easy to see that a flow that is feasible in the transaction graph is feasible also in the standard graph after the transformation and vice versa. In the transformation the number of the vertices is grown by a factor of $T1$ which might not be polynomial. It is possible to reduce the number of the vertices of G' to $O(|E|)$. There are $|E|$ edges between different layers in G'. Each such an edge touches two vertices . Vertices that are not touched by any of these edges are unnecessary and can be removed. The edges from the predecessor of the removed vertex to it and from it to its successor is replaced by an edge from the predecessor vertex to the successor vertex.

6 The Min Necessary Edge Problem

In the min necessary edge problem the goal is to find an edge with a minimum cost whose removal reduces the maximum flow between two given nodes, a source node s and a target node t. We first introduce some background material that will help us to design algorithms to the problem.

6.1 Background

In the max flow problem, the flow is computed between two specific nodes. Sometimes we want to determine the maximum flow between more than just one pair of nodes in the graph. A way to do it is by using the *all pair min cut problem*. In the *all pair min cut problem* we are required to find the minimum cuts between all pairs of nodes. The problem can be solved trivially using $O(n^2)$ calls to a max flow procedure. For undirected graphs, this problem can be solved more efficiently using *Gomory-Hu trees*. It turns out that among all $\binom{n}{2}$ pairs of vertices there are only at most $n-1$ distinct values for all the min cuts. In addition, it is possible to compute all these cuts using $n-1$ calls to max flow procedure [10] [11]. However for the directed case the property that there are only $n-1$ distinct values of cut values does not hold. An example for such a directed graph is the following [9]:



Let A be a matrix such that its entry a_{ij} is the value of minimum $i-j$ cut for $i \neq j$, and a_{ii} is arbitrary defined to ∞. For that above example:

$$A = \begin{pmatrix} \infty & 3 & 2 & 1 \\ 4^4+2 & \infty & 2 & 1 \\ 2\cdot 4^3+1 & 4^3+1+3 & \infty & 1 \\ 3\cdot 4^2 & 2\cdot 4^2+3 & 4^2+2 & \infty \end{pmatrix}$$

This example can be generalized to a graph with any number of vertices n. Above the diagonal there will be $n-1$ distinct values and below the diagonal all the values will be distinct, hence there will be $(n-1)+\frac{n(n-1)}{2}$ distinct values.

The standard value of a cut is the sum of the capacities of the edges that cross the cut. Cheng and Hu [6] showed that for undirected graphs, given an oracle access for some arbitrary function that assigns a value for each cut, it is possible to compute the min cut between all $\binom{n}{2}$ pairs of nodes according to this function by using $n-1$ oracle queries. An oracle query gets as an input two nodes and returns the minimum value of a cut that separate these two nodes. As pointed out by Benczúr [3], given a directed graph we can use the algorithm of Cheng and Hu to calculate the function $min\{F_{uv}, F_{vu}\}$ where F_{uv} is the min $u-v$ cut and F_{vu} is the min $v-u$ cut.

Theorem 11 *Let $G=(V,E)$ be a directed graph with a nonnegative capacity c_e associated with each edge $e \in E$. For all pairs of nodes u, v we can compute the minimum value of the maximum flow value from u to v and the maximum flow value from v to u using only $2(n-1)$ computations of maximum flow.*

6.2 Trivial algorithm

The trivial solution is to iterate over all $e \in E$ and check whether the max flow between s and t in $G' = (V, E-\{e\})$ is less than $|f|$. If it less than $|f|$ then e is a necessary edge. This trivial algorithm finds all necessary edges using m invocations to a max flow procedure. Then it can choose the edge with the minimum cost among all the necessary edges. In the next section we will show an algorithm which finds all the necessary edges using only $O(n)$ invocations to a max flow procedure.

Each edge that is a part of a min cut is a necessary edge since removing it decrease the min cut value, thus decreasing also the max flow value. By a single run of a max flow algorithm we obtain a maximum flow f and the corresponding residual graph G_f.

From G_f we can find one min cut $(S, \overline{S})$ by scanning the connected vertices to s. By this way we find necessary edges which are part of one min cut. However, there can be many more min cuts. Finding all the edges that are part of a min cut is not much harder than finding a min cut as it can also be done by a single invocation of a max flow procedure as in [15]. However, finding all the min cut edges is not enough since there could be other necessary edges which are not part of any min cut. The only immediate edges that are not suspected to be necessary ones are those with a flow of zero in f since removing them does change f. For each other edge $e \in E$, e is necessary if and only if there is cut that contains e with a value of less than $|f| + c_e$.

6.3 Finding All Necessary Edges using $O(n)$ Max flow computations

Our algorithm begins with one invocation to a max flow procedure in order to find a feasible maximum $s - t$ flow f and to obtain the residual graph $G_f = (V, E_f)$. Let f_{uv} be the flow of edge $(u, v) \in E$. The following theorem links between the condition that an edge (u, v) is necessary in G and the maximum flow between u and v in the residual graph G_f.

Theorem 12 *Let f be a maximum flow between s and t in $G = (V, E)$. Then $(u, v) \in E$ is a necessary edge if and only if the maximum flow between u and v in the residual graph $G_f^{uv} = \{V, E_f - (u, v)\}$ is less than f_{uv}.*

Proof Assume that the maximum flow between u and v in G_f^{uv} is at least f_{uv}. Then there exists a feasible flow f' of value f_{uv} in G_f^{uv}. We define a flow $\hat{f}$ as follows:

$$\hat{f}_e = \begin{cases} 0 & e = (u, v) \vee (v, u) \\ f_e + f'_e & o.w \end{cases}$$

It is easy to see that $\hat{f}$ is a feasible flow from s to t in G with a value of $|f|$. It follows that (u, v) is not a necessary edge.

For the other direction assume that (u, v) is not a necessary edge, we will build a feasible flow $\hat{f}$ in G_f^{uv} that has a value of f_{uv}. Let f' be a maximum flow between s and t in $G^{uv} = (V, E - (u, v))$. $|f'| = |f|$ since we assumed that (u, v) is not a necessary edge.

We define a flow $\hat{f}$ as follows:

$$\hat{f}_e = \begin{cases} 0 & e = (u, v) \\ f'_e - f_e & o.w \end{cases}$$

To prove that $\hat{f}_e$ is a feasible flow it is enough to show that the capacity constraints and flow conservation conditions hold for $\hat{f}_e$.

1. Capacity constraints: for each forward edge $e \neq (u, v)$ in G_f^{uv} it holds that $0 \leq f'_e \leq c_e$ since f' is a feasible flow in G^{uv}, by subtracting f_e we get that $-f_e \leq f'_e - f_e \leq c_e - f_e$ and thus $-f_e \leq \hat{f}_e \leq c_e - f_e$. By switching the direction of e, for each backward edge it holds that $\hat{f}_e \leq f_e$.

2. Flow conservation: for each vertex w such that $w \notin \{s, t, u, v\}$ it is clear that the flow conservation constraints holds since $\hat{f}$ is a subtraction of two feasible flows. For s and t the flow conservation constraints holds since $|f| = |f'|$.

The net flow of $\hat{f}$ is f_{uv} since

$$\begin{aligned} \sum_{u:(u,w)\in E_f^{uv}} \hat{f}_{uw} - \sum_{u:(w,u)\in E_f^{uv}} \hat{f}_{wu} &= f_{uv} + \sum_{u:(u,w)\in E_f} (f'_{uw} - f_{uw}) - \sum_{u:(w,u)\in_f} (f'_{wu} - f_{wu}) \\ &= f_{uv} + \Big(\sum_{u:(u,w)\in E_f} f'_{uw} - \sum_{u:(w,u)\in_f} f'_{wu} \Big) \\ &\quad - \Big(\sum_{u:(u,w)\in E_f} f_{uw} - \sum_{u:(w,u)\in_f} f_{wu} \Big) \\ &= f_{uv} \end{aligned}$$

The first equality holds because $\hat{f}_{uv} = 0$ while $f'_{uv} - f_{uv} = -f_{uv}$, and the last equality is due to the flow conservation of f and f'. $\square$

Note that in the above definition of G_f^{uv} if the edge (u, v) is saturated, i.e. $f_{uv} = c_{uv}$ then G_f^{uv} and G_f are the same graph since the edge (u, v) has capacity of zero in G_f. Given a maximum flow f, each edge $e \in E$ can be classified into one of the three categories based on the value of the flow on the edge f_e and its capacity c_e:

- $f_e = 0$: in this case e is not necessary.

- $f_e = c_e$: we can determine if e is necessary or not by a single max flow computation in G_f.

- $0 < f_e < c_e$: we can determine if e is necessary or not by a single max flow computation in G_f^e which is different for each e.

If $f_e = 0$ or $f_e = c_e$ we say that e is *restricted* and otherwise it is *free*. We wish to find a maximum flow with the least *free* edges since they are harder to deal with. We use a *cycle free solution* [1] in order to find such a flow.

Definition 13 *A flow f is called a cycle free solution if the network does not contain a cycle composed only of free arcs.*

A maximum flow f can be transformed into a cycle free maximum flow f'. While there are more than n free edges, we scan the first n free edges each time and find a cycle which is constructed form these free edges. Then we augment the flow in this cycle in an arbitrary direction until at least one edge becomes restricted. After at most m iterations we will remain with a cycle free solution since in each iteration one edge becomes restricted. Since there is no cycle that contains only free edges, there are at most $n - 1$ such edges. The total runtime of this transformation is $O(nm)$ since each iteration of finding a cycle and augmenting flow across this cycle takes $O(n)$ time.

In our algorithm we deal with free edges by invoking a max flow procedure for each such edge. The edges with flow of zero are not necessary ones. Then we are left to deal with the saturated edges (there can be $m - n + 1$ such edges).

For each saturated edge (u, v), if the max flow from u to v is less than f_{uv} then (u, v) is a necessary edge. Note that since (u, v) is saturated there is a backward edge (v, u) with a capacity of f_{uv} in the residual graph G_f. Therefore the max flow from v to u in G_f is at least f_{uv} which implies that if $min\{F_{uv}, F_{vu}\} < f_{uv}$ it must be that $F_{uv} < f_{uv}$ and (u, v) is necessary edge, and if $min\{F_{uv}, F_{vu}\} \geq f_{uv}$ then $F_{uv} \geq f_{uv}$ and (u, v) is not a necessary edge. By Theorem 11 the function $min\{F_{uv}, F_{vu}\}$ can be computed for each pair of nodes in V by using only $2(n - 1)$ computations of maximum flow. The full description of algorithm is as follows:

Algorithm

Input: A directed $G = (V, E)$ with a capacity c_e for each edge, and two nodes s and t with a max flow $|f|$ between them.
Output: A list of all necessary edges.

1. Find a maximum flow f from s to t in G.
2. Transform f into a cycle free flow.
3. For each free edge $(u, v) \in E$
   (a) Find a maximum flow f' from u to v in G_f^{uv}.
   (b) Add (u, v) to the necessary edge list if $f' < f_{uv}$.
4. Use Theorem 11 to compute $min\{F_{uv}, F_{vu}\}$ for each pair $u, v \in V$ in G_f.
5. For each saturated edge $(u, v) \in E$.
   (a) Add (u, v) to the necessary edge list if $min\{F_{uv}, F_{vu}\} < f_{uv}$.

The algorithm consists of $O(n)$ calls to a max flow procedure and finding a cycle free solution. The transformation into a cycle free solution takes $O(mn)$ time. There is no hope for a max flow algorithm to run in less than linear time of its input, and hence the total runtime of our algorithm equals to $O(n)$ invocations of max flow procedures.

7 The Flow Reduction Problem

We first show that the problem is NP-hard, and then we will develop two approximation algorithms for it.

7.1 NP hardness

st-k-FR is a minimization problem. Similarly, we can define a decision version, in which we are given also a positive number R and ask if there is a subset that reduces the maximum flow below a given threshold and has a removal cost of not more than R.

We show that the problem is NP-hard through two reduction. The first reduction is a natural one from *knapsack* which is NP-hard for the general case but has a pseudo-

polynomial algorithm to solve it. In the second reduction we show that also if the costs of the edges are bounded by some polynomial the problem is still NP-hard. The second reduction is from the *Clique Problem*. Similar reductions appeared previously in [16].

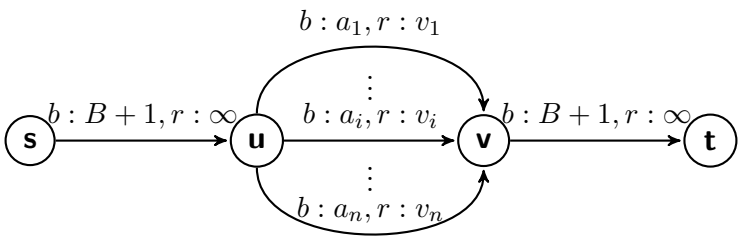
Reduction from Knapsack

The knapsack problem is defined as follows:

Definition 14 *Given a finite set U, a size $a_u \in Z^+$ and a value $v_u \in Z^+$ for each $u \in U$, and positive integers B and K, is there a subset $U' \subseteq U$ such that $\sum_{u \in U'} a_u \leq B$ and $\sum_{u \in U'} v_u \geq K$.*

Given input for knapsack we build a graph $G = (V, E)$ such that $V = \{s, t, u, v\}$ and there are edge between s and u and v to t both with capacity of $B+1$ and infinite cost. An infinite cost can be modeled by high cost such that there cannot be a minimum solution with such an edge. In this construction each such an edge can have a cost of $1 + \sum_{u \in U} v_u$. In addition, we add an edge between u and v for each $u \in U$. Thus there are n parallel edges between u and v, one for each item. Each such an edge has a capacity a_i and a cost of v_i that corresponds to the item it represents. The maximum flow in G between s and t is $B+1$ (assuming that $\sum_{u \in U} a_u > B$) and the flow threshold is set to B, thus a solution needs to reduce the maximum flow by at least one unit.

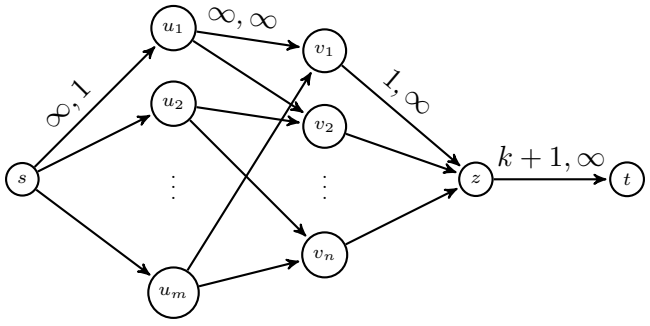
In order to reduce the flow between s and t we need to remove some of the parallel edges such that the cut $(\{s, u\}, \{v, t\})$ will have a capacity less than $B + 1$. Finding a subset with a minimum cost to remove from that cut is equivalent to finding a subset with a maximum cost to remain in the cut. It means that there exists a solution $U' \subseteq U$ to the knapsack problem if and only if there exists a subset $E' \subseteq E$ such that $\sum_{e \in E'} r_e \leq \sum_{u \in U} v_u - K$ and removing E' reduces the flow.



This reduction captures the property that finding subset of edges with a minimum cost for a specific cut is equivalent to solve a knapsack problem for this cut. The general case for our problem seems to be much harder since we have an exponential number of cuts.

Reduction from Clique

The input for the clique problem is an undirected graph $G = (V, E)$ and a number k, and the output is a clique of size k if exists. A clique of size k contains $\binom{k}{2}$ edges. Given an input for a clique problem $G = (V, E)$ with m edges and n vertices and a number k we build a network graph G' as follows. We create two layers, the first one consists of m vertices $\{u_1, .., u_m\}$, one for each edge of G, and the second layer consists of n vertices $\{v_1, .., v_n\}$, one for each vertex of G. We add two edges from each first layer vertex u_i that represents an edge $(v_j, v_k) \in E$ to these vertices in the second layer, thus we have the edges (u_i, v_j) and (u_i, v_k). Each such an edge has both infinite capacity and infinite cost. The source vertex s is connected to the first layer vertices, thus there is an edge (s, u_i) for each $1 \leq i \leq m$ with an infinite capacity and a removal cost of one. We add also an edge from each second layer vertex into another vertex z, thus there is an edge (v_i, z) for each $1 \leq i \leq m$ with capacity of one and infinite removal cost. Finally, z is connected to the sink t with an edge with an infinite removal cost and capacity of $K + 1$. The full construction is described in the following figure.



The maximum flow in G' from s to t is $K+1$. The only edges that have a removal cost of one and not of infinity are the ones from s to the first layer vertices, and hence these edges are the only candidates to be in the minimum subset. In order to reduce the flow from $K+1$ to at most K with minimum cost we need to remove the minimum number

of edges such that not more than K vertices in the second layer will be connected to first layer vertices whose edges from s are not removed. Therefore, if there is a clique of size k in G with set of edges E^k we can remove all the edges (s, u_i) such that $u_i \notin E^k$. All the second layer vertices that does not correspond to vertices in the clique in G will be disconnected from the first layer. Hence, The flow is reduced to K with a removal cost of $m - \binom{k}{2}$. If there is no clique of size k in G then each subset of $\binom{k}{2}$ edges in G touches more than k vertices and it follows that there is no subset of edges with removal cost at most $m - \binom{k}{2}$ that removing it from G' reduces the flow.

7.2 Approximation Algorithms

After proving that the problem is NP-hard it is convenient to look for approximation algorithms. We begin with a simple m-approximation greedy algorithm.

A Greedy Algorithm

The following greedy algorithm is a simple and quite trivial one. The algorithm starts by sorting all the edges according to their removal costs. Then it starts picking all the edges for the lowest cost one by one until the maximum flow is decreased to at most k.

Algorithm

1. Sort the edges according to their removal costs. Denote the sorted list by $\{e'_1, .., e'_m\}$.
2. $i = 0$
3. $S = \{e'_1\}$
4. While the maximum st flow in $G = (V, E - S)$ has a value of bigger or equal to k.
   (a) $i = i + 1$
   (b) $S = S \cup \{e'_i\}$
5. Output S.

Clearly the algorithm outputs a feasible solution. In addition, the edge with the maxi-

mum removal cost in the algorithm's solution does not have a bigger removal cost than the maximum removal cost edge in the optimal solution. Therefore:

$$C(ALG) = \sum_{i \in S} r_i \leq |S| \cdot max\{r_i | i \in S\} \leq |S| \cdot C(OPT)$$

As S contains at most m edges the algorithm achieve approximation ratio of m. The runtime of this algorithm is $O(m)$ max flows procedure calls.

Note that it is possible to improve the approximation ratio by a constant factor c while increasing the runtime to a power of c of the first algorithm. The algorithm first guesses (which is done by iterating on all the possibilities) the c edges with the highest removal cost for the optimal solution and then picks the other edges according to the above greedy algorithm. Let $r_1, \cdots, r_i$ be the edges the algorithm outputs sorted in decreasing cost. The algorithm has approximation guarantee of $\frac{m}{c}$ since:

$$C(ALG) = \sum_{i \in S} r_i \leq \sum_{i=1}^{c} r_i + (|S| - c) r_c \leq C(OPT) + (|S| - c) r_c =$$
$$C(OPT) + \frac{|S| - c}{c} \cdot c \cdot r_c \leq C(OPT) + \frac{|S| - c}{c} \cdot C(OPT) \leq \frac{m}{c} C(OPT)$$

Linear Programming

A minimal feasible solution to the *st-k-FR* problem identifies a cut. A solution removes edges from a cut such that the *residual capacity*, which is the capacity of the min-cut after removing the edges from the graph, decreases to at most k. The special case that $k = 0$ is the standard min-cut problem with respect to the edge removal cost. An Integer Program for the min-cut problem with a removal cost r_e for each edge is as follows.

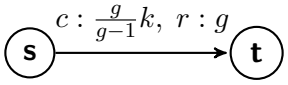
$$\begin{aligned} \min & \sum_{e \in E} r_e x_e \\ \text{s.t.} \quad & x_{uv} = d_v - d_u \quad \forall (u, v) \in E \\ & d_s = 0,\ d_t = 1 \\ & d_v \in \{0, 1\} \quad \forall v \in V \\ & x_e \in \{0, 1\} \quad \forall e \in E \end{aligned}$$

Similarly, for a general k, *st-k-FR* can be formulated by the following Integer Program:

$$\begin{aligned}
&\min \sum_{e \in E} r_e x_e \\
\text{s.t.} \quad & x_{uv} + y_{uv} = d_v - d_u \quad \forall (u,v) \in E && (1)\\
& \sum_{e \in E} c_e y_e \leq k && (2)\\
& d_s = 0,\ d_t = 1 \\
& d_v \in \{0,1\} \quad \forall v \in V && (3)\\
& x_e, y_e \in \{0,1\} \quad \forall e \in E && (4)
\end{aligned}$$

A solution for *st-k-FR* corresponds to a feasible solution for the IP and vice versa. Let $\mathcal{A}$ be a feasible solution of an instance of *st-k-FR* which is a set of removed edges. For each $e \in \mathcal{A}$ the corresponding variable x_e is set to one. Let $G' = (V, E - \mathcal{A})$ be the graph after removing $\mathcal{A}$ from G, then the maximum flow in G' from s to t is at most k as $\mathcal{A}$ is a feasible solution. Therefore, the min st-cut in G' is also at most k. Let $C(S,T)$ be the min cut in G'. The values for the vertex variables are set according to the partition, for each $v \in V$, $d_v = 0$ if $v \in S$, and $d_v = 1$ if $v \in T$. For each $e \in C(S,T)$ the variable y_e is set to one. As $c(S,T) \leq k$ in G', the residual capacity constraint (2) holds. In addition, for each edge $e \in C(S,T)$ in G either $x_e = 1$ or $y_e = 1$, and hence constraint (1) holds. Similarly, given a feasible solution to the IP it can be transformed into a feasible solution for *st-k-FR* with the same value by removing an edge e if $x_e = 1$.

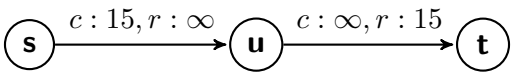
To transform the Integer Program to a Linear Program constraint (3) is relaxed to $0 \leq d_v \leq 1$, and constraint (4) is relaxed to $0 \leq x_e, y_e \leq 1$. However, the integrality gap of the linear program is large. The following example has integrality gap g for any given $g > 0$. There are two vertices s and t connected by one edge with capacity $\frac{g}{g-1}k$, and removal cost of g, and the maximum flow has to be decreased to k.



An optimal integral solution must remove the edge ($y_e = 0,\ x_e = 1$), and hence has a removal cost of g. An optimal fractional solution sets $y_e = \frac{g-1}{g}$ and $x_e = \frac{1}{g}$. The residual capacity constraint (2) holds as $\sum_{e \in E} c_e y_e = \frac{g-1}{g} \cdot \frac{g}{g-1} k = k$, and constraint (1)

holds as $x_e + y_e = \frac{1}{g} + \frac{g-1}{g} = 1$. Hence, the fractional solution has a removal cost of $r_e x_e = \frac{1}{g} \cdot g = 1$, and the integrality gap is g.

Another example which shows that a fractional solution with fractional vertices has a better removal cost than an integral solution is the following:



The flow has to be reduced to at most $k = 10$. An optimal integral solution removes the edge (u, t), and has a removal cost of 15. A fractional solution sets $d_u = \frac{2}{3}$, $x_{su} = 0$, $y_{su} = \frac{2}{3}$, $x_{ut} = \frac{1}{3}$, and $y_{su} = 0$, and hence the removal cost $r_{ut} x_{ut} = \frac{1}{3} \cdot 15 = 5$.

Bi-Criteria approximation

As the integrality gap is large, a standard rounding technique of the LP relaxation will not achieve a decent approximation guarantee. Instead, we design a bi-criteria approximation. The idea of our bi-criteria approximation is to decompose an optimal solution of the LP into a convex combination of integral solutions. Some of the integral solutions might not be feasible solutions as they will violate the residual capacity constraint. Nevertheless, as the weighted average of the integral solutions equals to the optimal solution of the LP, one of the integral solutions will be a good approximate solution.

Theorem 15 *Let X' be a feasible solution of the LP for st-k-FR. Then X' can be decomposed into a convex combination of at most $m + n$ integral solutions. In other words, there exists $p_1, p_2, \ldots, p_l > 0$, and integral solutions (which might violate the residual capacity constraint) $X^{(1)}, X^{(2)}, \ldots, X^{(l)}$ that satisfy:*

$$p_1 X^{(1)} + p_2 X^{(2)} + \ldots + p_l X^{(l)} = X'$$
$$p_1 + p_2 + \ldots + p_l = 1$$

for $l \leq m + n$.

Proof Let $\alpha_1 < \ldots < \alpha_r$ be all values given in X' to the vertex variables d_v. There are at most n different values as each value corresponds to at least one vertex. In addition,

$\alpha_1 = 0$ as $d_s = 0$, and $\alpha_r = 1$ as $d_t = 1$.

In order to decompose the fractional solution into a convex combination of integral solutions, we show a randomized scheme to choose an integral solution which in expectation equals to X'. The integral solution $X^{(i)}$ is constructed according to α which is chosen uniformly over $[0, 1]$. α partitions the vertices into two sets (S_i, T_i):

1. $S_i = \{v \mid d_v \leq \alpha\}$

2. $T_i = \{v \mid d_v > \alpha\}$

For each $v \in S_i$, $d_v^{(i)} = 0$, and for each $v \in T_i$, $d_v^{(i)} = 1$ in X^i. Then for each $v \in V$:

$$\mathbf{E}_{\alpha \sim U(0,1)} \left[d_v^{(i)} \right] = 1 \cdot \Pr_{\alpha \sim U(0,1)} \left[d_v^{(i)} = 1 \right] = \Pr_{\alpha \sim U(0,1)} \left[d_v > \alpha \right] = d_v$$

In addition, for each $(u, v) \in E$, $x_{uv}^{(i)} = 1$ if $\alpha \in [d_u, d_u + x_{uv}]$, and $y_{uv}^{(i)} = 1$ if $\alpha \in (d_u + x_{uv}, d_u + x_{uv} + y_{uv}]$. Note that $d_u + x_{uv} + y_{uv} = d_v$ since $x_{uv} + y_{uv} = d_v - d_u$. Then:

$$\mathbf{E}_{\alpha \sim U(0,1)} \left[x_{uv}^{(i)} \right] = 1 \cdot \Pr_{\alpha \sim U(0,1)} \left[x_{uv}^{(i)} = 1 \right] = \Pr_{\alpha \sim U(0,1)} \left[d_u \leq \alpha \leq d_u + x_{uv} \right] = x_{uv}$$
$$\mathbf{E}_{\alpha \sim U(0,1)} \left[y_{uv}^{(i)} \right] = 1 \cdot \Pr_{\alpha \sim U(0,1)} \left[y x_{uv}^{(i)} = 1 \right] = \Pr_{\alpha \sim U(0,1)} \left[d_u + x_{uv} < \alpha \leq d_u + x_{uv} + y_{uv} \right] = y_{uv}$$

For each edge (u, v) in the cut (S_i, T_i) either $x_{uv}^{(i)} = 1$ or $y_{uv}^{(i)} = 1$ and the constraint $x_{uv}^{(i)} + y_{uv}^{(i)} = d_v - d_u$ holds. Therefore, the only constraint that might be violated for $X^{(i)}$ is the residual capacity constraint. As the expectation of each variable in $X^{(i)}$ equals to its corresponding variable in X' then:

$$\mathbf{E}_{\alpha \sim U(0,1)}[X^{(i)}] = X' \tag{5}$$

The number of different integral solutions in the probability distribution is at most $m + n$. The vertex variables split the interval $[0, 1]$ into at most $n - 1$ intervals, such that if α is in different interval the partition of the vertices is different. In addition, each edge splits at most one interval into two intervals for which the solution of the two intervals becomes different. The splitting point of the interval is when $d_u + x_{uv} = d_v - y_{uv}$ for an edge (u, v). Hence, there are not more than $m+n$ different intervals, and therefore

there exists $p_1, p_2, \ldots, p_l > 0$ and integral solutions $X^{(1)}, X^{(2)}, \ldots, X^{(l)}$ such that:

$$\mathbf{E}_{\alpha \sim U(0,1)}[X^{(I)}] = p_1 X^{(1)} + p_2 X^{(2)} + \ldots + p_l X^{(l)}$$
$$p_1 + p_2 + \ldots + p_l = 1$$

for $l \leq n + m$. Combining the above with (5), the theorem holds. □

Denote by R', k' the total removal cost and the residual capacity of a feasible solutions X' respectively, then by the last theorem and linearity of expectation, X' can be decomposed into a convex combination of integral solutions $X^{(1)}, X^{(2)}, \ldots, X^{(l)}$ which satisfy:

$$p_1 R^{(1)} + p_2 R^{(2)} + \ldots + p_l R^{(l)} = R'$$
$$p_1 k^{(1)} + p_2 k^{(2)} + \ldots + p_l k^{(l)} = k'$$

where $R^{(i)}$ is the removal cost of solution $X^{(i)}$, and $k^{(i)}$ is its residual capacity. An optimal solution of the LP has a special property that there are two integral solutions which a convex combination of their removal cost and residual capacity equals to the removal cost and the residual capacity of the optimal solution.

Theorem 16 *Let X^* be an optimal solution of the LP. Then there are two integral solutions solutions $X^{(1)}, X^{(2)}$ which satisfy:*

$$\lambda_1 R^{(1)} + \lambda_2 R^{(2)} = R^*$$
$$\lambda_1 k^{(1)} + \lambda_2 k^{(2)} = k^*$$

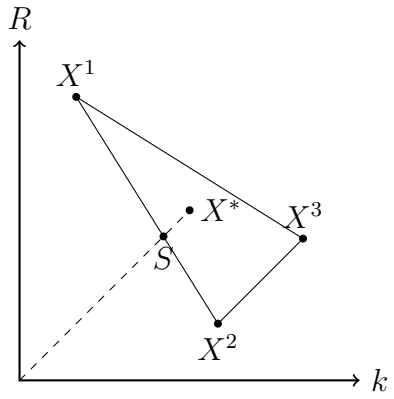
for some $\lambda_1 + \lambda_2 = 1$ and $\lambda_1, \lambda_2 \geq 0$.

Proof We give a geometric proof. By Theorem 15, X^* is decomposed into a convex combination of $l \leq n+m$ integral solutions. If $l \leq 2$ then X^* is already decomposed into at most two solutions, and we are done. Hence we can assume that $l > 2$. Each solution $X^{(i)}$ can be represented on the plane by its removal cost and its residual capacity. If all the l points are on a single line, then two of them can be chosen, and X^* can be decomposed to a convex combination of these two solutions. We will show that the other case, which is that there are at least three solutions that are not on the same line, leads to a contradiction. As there is a convex combination with coefficients $p_1, p_2, \ldots, p_l$

that satisfies:

$$p_1 R^{(1)} + p_2 R^{(2)} + \ldots + p_l R^{(l)} = R^*$$
$$p_1 k^{(1)} + p_2 k^{(2)} + \ldots + p_l k^{(l)} = k^*$$

and not all the points are on one line, the point (k^*, R^*) is strictly contained in the convex hull of $(k^{(i)}, R^{(i)})$ for $1 \leq i \leq l$. Let $S = (k_s, R_s)$ be the intersection point of the convex hull with the line that passes through $(0, 0)$ and (k^*, R^*). S is a feasible solution of the LP as it is a convex combination of two integral solutions and $k_s \leq k^*$. It also has a better removal cost than X^* since $R_s < R^*$, which is a contradiction for the optimality of X^*.



□

Next we show that for every $\epsilon > 0$, a simple algorithm which outputs one of the integral solutions from the convex combination of the optimal solution achieves $((1+\epsilon) \vee (1+\frac{1}{\epsilon}))$ approximation guarantee. Below is a description of the approximation algorithm:

Algorithm

Input: An instance of *st-k-FR*, and $\epsilon > 0$.
Output: A solution with $(\ (1+\epsilon)\ \vee\ (1+\frac{1}{\epsilon})\ )$ approximation guarantee.

1. Solve the LP of *st-k-FR*, and denote the optimal solution by X^*.
2. By Theorems 15 and 16, obtain two integral solutions solutions $X^{(1)}, X^{(2)}$ which satisfy:
$$\lambda_1 R^{(1)} + \lambda_2 R^{(2)} = R^*$$
$$\lambda_1 k^{(1)} + \lambda_2 k^{(2)} = k^*$$
for some $\lambda_1 + \lambda_2 = 1$ and $\lambda_1, \lambda_2 \geq 0$.
3. If $R^{(1)} \leq (1+\epsilon)R^*$ and $k^{(1)} \leq k^*$ output $X^{(1)}$, and otherwise output $X^{(2)}$.

Proof (Theorem 6). By Theorem 15, X^* can be decomposed into a convex combination of $X^{(1)}, X^{(2)}, .., X^{(l)}$ with coefficients $p_1, p_2, \ldots, p_l$ that satisfy:

$$\begin{aligned} p_1 R^{(1)} + p_2 R^{(2)} + \ldots + p_l R^{(l)} &= R^* \\ p_1 k^{(1)} + p_2 k^{(2)} + \ldots + p_l k^{(l)} &= k^* \end{aligned} \tag{1}$$

Theorem 16 implies we can assume that $l = 2$, though this is not needed in the subsequent proof. By applying a standard averaging argument on (1), there is an integral solution $X^{(i)}$ with removal cost of at most R^*. However, we cannot have a better bound than $\frac{k^*}{p_i}$ on the residual capacity of that $X^{(i)}$. In order to bound both parameters simultaneously, the removal cost and the residual capacity are unified by summing a weighted combination of the two equations. For any $\beta \geq 0$, it holds that:

$$p_1(R^{(1)} + \beta k^{(1)}) + p_2(R^{(2)} + \beta k^{(2)}) + \ldots + p_l(R^{(l)} + \beta k^{(l)}) = R + \beta k$$

By averaging there exists a solution $X^{(i)}$ that satisfies:

$$R^{(i)} + \beta k^{(i)} \leq R^* + \beta k^* \tag{2}$$

Note that a different choice of β might require a different choice of $X^{(i)}$. By (2), either $R^{(i)} \leq R^*$ or $k^{(i)} \leq k^*$. Given $\epsilon > 0$, β is chosen such that:

$$\epsilon R^* = \beta k^* \tag{3}$$

If $R^{(i)} \geq R^*$ then:

$$R^{(i)} \leq R^* + \beta k^* = R^* + \epsilon R^* = (1+\epsilon)R^*$$

If $k^{(i)} \geq k^*$ then:

$$\beta k^{(i)} \leq R^* + \beta k^* = \frac{\beta k^*}{\epsilon} + \beta k^* = (1+\frac{1}{\epsilon})\beta k^*$$

and $k^{(i)} \leq (1+\frac{1}{\epsilon})k^*$. □

Note that the algorithm has even a stronger approximation guarantee than ($(1+\epsilon) \vee (1+\frac{1}{\epsilon})$). For example consider the case that $\epsilon = 1$ which gives an approximation of $(2 \vee 2)$. By (3), $R^* = \beta k^*$ and inequality (2) becomes $R^{(i)} + \beta k^{(i)} \leq 2R^*$. Suppose that $R^{(i)} = 2R^*$, then it must be that $k^{(i)} = 0$, which is tighter than the $k^{(i)} \leq k^*$ guarantee provided by the $(2 \vee 2)$ approximation. Equation (2) defines the tradeoff between the approximation guarantees of the removal cost and the residual capacity.

A Parametric Search Algorithm

Another way of getting a ($(1+\epsilon) \vee (1+\frac{1}{\epsilon})$) bicriteria approximation for *st-k-FR* is by using a reduction to min-cut. This was apparently referred to in [4] without providing details (and while mentioning the term parametric search and providing references to unpublished work of Rao, Shmoys and Tardos [5]). This was also the approach taken in [7] in the part of their paper that dealt with a restricted version of *st-k-FR* (in which all capacities are 1). We review this approach and compare it to ours.

Assume that the optimal solution for the *st-k-FR* instance uses removal cost R^* and achieves a residual capacity of k^*. Suppose for simplicity that the value of the ratio $\frac{R^*}{k^*}$ is given. Scale the removal cost of each edge in the input instance by a multiplicative factor of $\frac{k^*}{R^*}$. This does not change the optimal solution, but it leads to the convenient situation in which for the scaled values of capacities, the optimal solution exactly balances between removal cost and residual capacity. Namely, now $R^* = k^*$.

We now show how to get a $(2 \vee 2)$ bicriteria approximation. For every edge e, give it a weight w_e equal to the minimum of its removal cost r_e and (scaled) capacity c_e. Now find a minimum s-t-cut with respect to this weight function, and let W denote its weight. Observe that necessarily $W \leq R^* + k^*$. We now modify the cut in order to get a solution to *st-k-FR*. For every edge of the cut, its weight corresponds either to its removal cost, in which case we remove it, or to its capacity, in which case we count this towards the residual capacity. Let R be the total removal cost obtained this way, and let C be the total residual capacity. Hence $R + C = W \leq R^* + k^*$. As $R^* = k^*$, either $C \leq k^*$ and $R \leq 2R^*$, or $R \leq R^*$ and $C \leq 2k^*$, implying a $(2 \vee 2)$ bicriteria approximation. The same approach, with a further scale of every capacity by a factor of ϵ, can be used to get a $((1+\epsilon) \vee (1+\frac{1}{\epsilon}))$ bicriteria approximation. Details are omitted. The advantage of the min-cut approach is that it only involves a min-cut computation, whereas the our randomized rounding method requires solving an LP.

The advantage of the randomized rounding method is that it produces a pair of solutions that are simultaneously good for all values of ϵ in the bicriteria approximation, whereas in the min-cut approach exploring new values of ϵ might require additional min-cut computations. The main difficulty with the min-cut approach is determining the ratio $\frac{R^*}{k^*}$ so as to know by how much to scale capacities. It is NP-hard to compute this ratio (because for $k^* = k$, computing R^* is NP-hard), and it is also difficult to estimate it within any constant factor (as there is no known algorithm that approximates R^*). However, one can systematically search over possible scaling factors for the capacity (in hope of hitting the ratio $\frac{R^*}{k^*}$). This involves iteratively performing min-cut computations with different scaling ratios. There are several approaches to keep the number of iterations relatively small, say, polylogarithmic in m. One may search only ratios that are powers of $1+\delta$ (for some small δ, if one is willing to tolerate a small change to ϵ). The number of iterations will be $\log_{1+\delta} m$ because one can approximate *st-k-FR* within a ratio of m. Alternatively, if one is not willing to tolerate the small change in ϵ, one can try parametric search which involves a binary search on the R^* (as we can assume that $k^* = k$). In any case, the outcome of the search is a sequence of solutions with values (R_i, k_i). If for some solution i we have $k \leq k_i \leq (1+\frac{1}{\epsilon})k$ then necessarily $R_i \leq R^*$, and solution i is a $((1+\epsilon) \vee (1+\frac{1}{\epsilon}))$ bicriteria approximation. If there is no i with $k \leq k_i \leq (1+\epsilon)k$ then we use the observation that when the scaling ratio $\epsilon\frac{k^*}{R^*}$ is hit (or approximated up to a $1+\delta$ factor), the value of k must have been at most k^* (because it is at most $(1+\frac{1}{\epsilon}k^*)$) and the value of R is at most $(1+\epsilon)R^*$.

Hence if one outputs the solution of smallest value of R_i among those with $R \leq R^*$, this is a $((1+\epsilon) \vee (1+\frac{1}{\epsilon}))$ bicriteria approximation.

Another way to obtain the ratio $\frac{R^*}{k^*}$ is solve the LP for *st-k-FR*, get from the solution a pair of values $C^* \leq k$ and R^*, and then use them in the min-cut approach. Hence now the min-cut approach does not replace the LP, but only provides a rounding technique for the LP.

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, and K. Weihe. Network flows: theory, algorithms and applications. *ZOR-Methods and Models of Operations Research*, 41(3):252–254, 1995.

[2] S. Barman and S. Chawla. Region growing for multi-route cuts. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 404–418. Society for Industrial and Applied Mathematics, 2010.

[3] A.A. Benczúr. Counterexamples for directed and node capacitated cut-trees. *SIAM Journal on Computing*, 24:505, 1995.

[4] C. Burch, R. Carr, S. Krumke, M. Marathe, C. Phillips, and E. Sundberg. A decomposition-based pseudoapproximation algorithm for network flow inhibition. *Network Interdiction and Stochastic Integer Programming*, pages 51–68, 2003.

[5] Carr R. D. Krumke S. O. Marathe M. Phillips C Burch, C and E Sundberg. Multicriteria approximation through decomposition. *(unpublished manuscript)*, 2001.

[6] Chung-Kuan Cheng and T. C. Hu. Ancestor tree for arbitrary multi-terminal cut functions. In *Proceedings of the 1st Integer Programming and Combinatorial Optimization Conference*, pages 115–127, Waterloo, Ont., Canada, 1990. University of Waterloo Press.

[7] J. Chuzhoy, Y. Makarychev, A. Vijayaraghavan, and Y. Zhou. Approximation algorithms and hardness of the k-route cut problem. In *Proceedings of the Twenty-Third*

Annual ACM-SIAM Symposium on Discrete Algorithms, pages 780–799. SIAM, 2012.

[8] L.R. Ford and D.R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956.

[9] H. Frank and I.T. Frisch. *Communication, transmission, and transportation networks*. Addison-Wesley, 1971.

[10] R.E. Gomory and T.C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

[11] D. Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, 19:143, 1990.

[12] Dan Gusfield. A graph theoretic approach to statistical data security. *SIAM J. Comput.*, 17(3):552–571, June 1988.

[13] W. Kishimoto. A method for obtaining the maximum multiroute flows in a network. *Networks*, 27(4):279–291, 1996.

[14] C.A. Phillips. The network inhibition problem. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 776–785. ACM, 1993.

[15] Jean-Claude Picard and Maurice Queyranne. On the structure of all minimum cuts in a network and applications. In *Combinatorial Optimization II*, volume 13 of *Mathematical Programming Studies*, pages 8–16. Springer Berlin Heidelberg, 1980.

[16] R.K. Wood. Deterministic network interdiction. *Mathematical and Computer Modelling*, 17(2):1–18, 1993.