Efficient Algorithms for Computing All Low s-t Edge Connectivities and Related Problems

Ramesh Hariharan* Telikepalli Kavitha† Debmalya Panigrahi‡

Abstract

Given an undirected unweighted graph $G = (V, E)$ and an integer $k \geq 1$, we consider the problem of computing the edge connectivities of all those (s, t) vertex pairs, whose edge connectivity is at most k. We present an algorithm with expected running time $\tilde{O}(m + nk^3)$ for this problem, where $|V| = n$ and $|E| = m$. Our output is a weighted tree $\mathcal{T}$ whose nodes are the sets $V_1, V_2, \ldots, V_\ell$ of a partition of V, with the property that the edge connectivity in G between any two vertices $s \in V_i$ and $t \in V_j$, for $i \neq j$, is equal to the weight of the lightest edge on the path between V_i and V_j in $\mathcal{T}$. Also, two vertices s and t belong to the same V_i for any i if and only if they have an edge connectivity greater than k. Currently, the best algorithm for this problem needs to compute all-pairs min-cuts in an $O(nk)$ edge graph; this takes $\tilde{O}(m + n^{5/2}k \min\{k^{1/2}, n^{1/6}\})$ time. Our algorithm is much faster for small values of k; in fact, it is faster whenever k is $o(n^{5/6})$.

Our algorithm yields the useful corollary that in $\tilde{O}(m + nc^3)$ time, where c is the size of the global min-cut, we can compute the edge connectivities of all those pairs of vertices whose edge connectivity is at most αc for some constant α. We also present an $\tilde{O}(m + n)$ Monte Carlo algorithm for the approximate version of this problem. This algorithm is applicable to weighted graphs as well. Our algorithm, with some modifications, also solves another problem called the minimum $\mathbb{T}$-cut problem. Given $\mathbb{T} \subseteq V$ of *even* cardinality, we present an $\tilde{O}(m + nk^3)$ algorithm to compute a minimum cut that splits $\mathbb{T}$ into two odd cardinality components, where k is the size of this cut.

1 Introduction

Let $G = (V, E)$ be an undirected unweighted graph with $|V| = n$ and $|E| = m$. The edge connectivity of two vertices $s, t \in V$ is defined as the size of the smallest cut that separates them; such a cut is called a *minimum s-t cut*. A classical result in graph connectivity, due to Gomory and Hu [6], states that the edge connectivities of all pairs of vertices in an undirected graph can be computed using $n - 1$ (rather than the naïve $\binom{n}{2}$) max-flow computations. Their algorithm computes a weighted cut-tree $\mathcal{T}$, known as the Gomory-Hu tree, on V, with the property that the edge connectivity between any two vertices s and t in the graph exactly equals the weight on the lightest edge in the unique s-t path in $\mathcal{T}$. Further, the partition of the vertices produced by removing this edge from $\mathcal{T}$ is a minimum s-t cut in the graph.

However, in many applications, the construction of the entire Gomory-Hu tree is redundant. For instance, one of the most popular applications of graph connectivity algorithms is in deciding the robustness of a network and finding the edge connectivities of the unreliable portions in it. In such situations, one is interested only in finding the edge connectivities of those pairs of vertices that are poorly connected in the network. In particular, we look at the following problem, which we call the "at most k-connectivity problem".

Given a graph G and an integer $k \geq 1$, compute the edge connectivities of all pairs of vertices in G whose edge connectivity is less than or equal to k.

The output should be represented succinctly as a weighted tree $\mathcal{T}'$ whose nodes are $V_1, V_2, \ldots, V_\ell$ (a partition of V) with the property that any two vertices s, t are in the same subset V_i, for any i, if and only if the s-t edge connectivity in G is greater than k and the edge connectivity of a vertex in V_i and a vertex in V_j for $i \neq j$, is equal to the weight of the lightest edge on the path between V_i and V_j in $\mathcal{T}'$.

1.1 Background and New Results.

Currently, the best algorithm to solve the at most k-connectivity problem involves computing all-pairs min-cuts. Thus *all* s-t edge connectivities are computed. This means that the entire Gomory-Hu tree is constructed, which involves $n-1$ max-flow computations [6, 8]. This leads to a time complexity of $O(nm \min\{m^{1/2}, n^{2/3}\} \log(n^2/m))$ using the Golberg-Rao max-flow algorithm [5]. This can be improved to $O(m + n^{5/2}k \min\{k^{1/2}, n^{1/6}\} \log(n/k))$ by preprocessing[1] the graph. Thus the previous best time complexity was $\tilde{O}(m + n^{5/2}k \min\{k^{1/2}, n^{1/6}\})$. We show the following result here.

*Strand Life Sciences, Bangalore, India; ramesh@strandls.com

†CSA Department, Indian Institute of Science, Bangalore, India; kavitha@csa.iisc.ernet.in

‡Bell Labs Research, Bangalore, India; pdebmalya@lucent.com. This work was done while the author was at the Indian Institute of Science, Bangalore.

[1] This preprocessing uses the Nagamochi-Ibaraki construction on unweighted graphs [10] to keep down the number of edges in the graph to $O(nk)$ without affecting the output of the at most k-connectivity problem.

THEOREM 1.1. *The at most k-connectivity problem can be solved in expected $\tilde{O}(m + nk^3)$ time in an undirected unweighted graph with n vertices and m edges.*

The parameter k is typically a small value, since we are after all interested in identifying *poorly* connected pairs of vertices. For small values of k, our algorithm is much faster than the previous best. In fact, our algorithm is faster whenever k is $o(n^{5/6})$. Recall that our algorithm needs to output a weighted tree $\mathcal{T}'$ and the edge connectivities of all poorly connected (s,t) vertex pairs can be easily computed from this tree $\mathcal{T}'$. A related problem is that of finding all k-edge connected components in a given graph; an $O(m+n^2k^2)$ algorithm was given in [11] for this problem in an undirected graph with m edges and n vertices.

The fastest algorithm for computing a global min-cut (i.e., smallest among all cuts) in a graph is due to Gabow [4] and it runs in $\tilde{O}(m + nc^2)$ time on undirected graphs, where c is the size of a global min-cut. However, the algorithm does not readily extend to computing edge connectivity between pairs of vertices whose edge connectivity is slightly greater than the global minimum. In particular, one might be interested in computing the edge connectivities of all those pairs of vertices (s,t) whose edge connectivity is within a factor α of the global min-cut c. Our result shows that we can compute all s-t edge connectivities which are at most αc in $\tilde{O}(m + n(\alpha c)^3)$ time.

Using a sampling technique due to Karger [9], we show the following theorem, which gives us an almost linear time algorithm for the approximate version of this problem, when α and ϵ are constants.

THEOREM 1.2. *All pairs of vertices whose edge connectivity is at most α times the size of a minimum cut of G and no pair whose edge connectivity is more than $(1+\epsilon)\alpha$ times the size of a minimum cut can be identified with probability $1 - 1/n$ in expected $\tilde{O}(m + n(\alpha/\epsilon^2)^3)$ time.*

This theorem applies to weighted graphs as well.

We also use a sampling technique due to Benczúr and Karger [2] in conjunction with our at most k-connectivity algorithm to devise an algorithm for finding the approximate edge connectivity between each pair of vertices in $\tilde{O}(n^{5/2})$ time, with the additional guarantee that if the edge connectivity between a pair of vertices is at most $\sqrt{n}$, then it is reported exactly.

Another problem we consider is a natural generalization of the minimum s-t cut problem called the *minimum $\mathbb{T}$-cut* problem:
Given a subset of vertices $\mathbb{T}$ of even cardinality, find the size of a smallest cut in the graph that splits $\mathbb{T}$ into two odd-cardinality components.

The minimum $\mathbb{T}$-cut problem arises in studying the perfect matching polytope of a graph [7]. Moreover, minimum $\mathbb{T}$-cut procedures are used in several branch and bound algorithms for problems like the TSP. The minimum $\mathbb{T}$-cut problem can be solved by constructing a Gomory-Hu tree. Alternatively, there is an algorithm by Rizzi [12] which does not explicitly construct the Gomory-Hu tree but uses at most $|\mathbb{T}| - 1$ max-flow computations, leading to a time complexity of $O(|\mathbb{T}| \cdot (\text{time for a max-flow in } G))$. Here we show the following result.

THEOREM 1.3. *The minimum $\mathbb{T}$-cut problem can be solved in expected $\tilde{O}(m + nk^3)$ time in an undirected unweighted graph with n vertices and m edges, where k is the size of a minimum $\mathbb{T}$-cut.*

1.2 Our Methods. Our algorithms are based on the notion of a *partial* Gomory-Hu tree. For any k, imagine contracting all those edges in the Gomory-Hu tree $\mathcal{T}$, whose weight is greater than k. This defines a tree $\mathcal{T}'$ on subsets of vertices, call them $V_1, V_2 \ldots, V_\ell$, where $s,t \in V_i$ for some i if and only if all the edges in the s-t path in the tree $\mathcal{T}$ have weight greater than k. Hence, s-t edge connectivity in G for any s,t in the same V_i, for any i, is at least $k+1$ and the s-t min-cut value for any pair s,t that belong to different sets V_i, V_j in the partition $V_1, V_2 \ldots, V_\ell$ is the weight of the lightest edge on the path between the nodes V_i and V_j in the tree $\mathcal{T}'$.

Clearly, computing a partial Gomory-Hu tree solves the at most k-connectivity problem. We obtain our efficient algorithm for computing a partial Gomory-Hu tree by using a *minimum Steiner cut* algorithm of Cole and Hariharan [3] as our basic subroutine. As we shall see, this offers multiple advantages over the traditional flow-based approach. However a naïve implementation of this algorithm as our basic subroutine would imply a running time of $\tilde{O}(m + n \cdot nk^3)$ for our algorithm. We use randomization here and show that if we choose the root of the trees built in the minimum Steiner cut algorithm uniformly at random from the Steiner set, then the expected running time of our entire algorithm to compute the partial Gomory-Hu tree is just $\tilde{O}(m + nk^3)$.

Organization of the paper. The rest of the paper is organized as follows. Section 2 discusses the Gomory-Hu tree construction algorithm and the minimum Steiner cut algorithm and shows how they can be used to give an algorithm for computing a partial Gomory-Hu tree. Section 3 shows how this algorithm can be made efficient by reusing the work done in various iterations of the minimum Steiner cut algorithm. Section 3.2 has our approximation algorithm for the αc-connectivity problem. Our algorithm for the minimum

$\mathbb{T}$-cut problem is given in Section 4. We conclude with a summary of the results presented in this paper.

2 Partial Gomory-Hu tree

In this section we review the Gomory-Hu tree construction algorithm [6, 8] and present a simple algorithm to construct a partial Gomory-Hu tree. The algorithm to construct a Gomory-Hu tree uses the classical concept of submodularity of cuts.

FACT 2.1. (SUBMODULARITY OF CUTS) *If A and B are two subsets of vertices in a graph and $\delta(X)$ represents the size of the cut $(X, V \setminus X)$, then $\delta(A)+\delta(B) \geq \delta(A \cap B) + \delta(A \cup B)$.*

Fact 2.1 leads to the following theorem, which is used in the algorithms in [6] and [8].

THEOREM 2.1. *If $(S, V \setminus S)$ is a minimum s-t cut in G and $u, v \in S$, then there exists a minimum u-v cut $(S^*, V \setminus S^*)$ such that $S^* \subset S$.*

The Gomory-Hu tree construction algorithm [6] initializes the cut-tree $\mathcal{T}$ to a single node that contains the entire vertex set. At any step of the algorithm, pick a node S of $\mathcal{T}$ containing more than one vertex and choose any two vertices s and t in S. Contract the entire subtree subtended at each neighbor of S into a single node and perform a max flow computation from s to t in the new graph. Theorem 2.1 ensures that the minimum s-t cut thus obtained (we call it C) is also a minimum s-t cut in the original graph. Now, in $\mathcal{T}$, the node S is split into S_1 and S_2 according to C and the two nodes thus formed are joined by an edge of weight equal to the size of C. Further, all the neighboring subtrees of S become neighboring subtrees of S_1 or S_2 depending upon which side of C they lie on. The algorithm terminates when all the nodes of $\mathcal{T}$ become singleton sets. Thus $\mathcal{T}$ is a weighted tree whose nodes are the vertices of V. It can be shown that $\mathcal{T}$ captures all-pairs min-cuts.

However, constructing the entire Gomory-Hu tree is not necessary for the at most k-connectivity problem. We would like to construct a partial Gomory-Hu tree which captures only those s-t min-cuts whose size is $\leq k$. The problem with a flow-based approach in such a scenario is that the values of the max flows in different iterations might be arbitrarily ordered and hence we would not know till the $(n-1)$th iteration if we have identified all s, t pairs whose edge connectivity is at most k.

To circumvent this problem, we introduce the *minimum Steiner cut* problem. Given a subset $S \subseteq V$, called the Steiner set, the *Steiner connectivity* is the size of the smallest cut that splits S into two non-empty components. Such a cut is called a *minimum Steiner cut*. The following simple observation about minimum Steiner cuts helps us in the at most k-connectivity problem.

FACT 2.2. *If S^* and S are subsets of vertices satisfying $S^* \subseteq S$, then the minimum Steiner cut of S^* is at least as large as the minimum Steiner cut of S. This follows from the fact that a Steiner cut of S^* is also a Steiner cut of S.*

Fact 2.2 implies that if the Steiner cut of a node in the cut-tree is larger than k, then all pairs of vertices in that node have an edge connectivity larger than k. We will use Fact 2.2 to design a simple deterministic algorithm (Algorithm 1.1) for computing a partial Gomory-Hu tree that solves the at most k-connectivity problem. This algorithm follows the same approach as the Gomory-Hu algorithm except for replacing the s-t min-cut algorithm with a minimum Steiner cut algorithm. It is easy to see that when this algorithm terminates, each node of $\mathcal{T}$ has minimum Steiner cut value larger than k. Thus our algorithm constructs a partial Gomory-Hu tree corresponding to the parameter k.

The fastest algorithm to compute a minimum Steiner cut is by Cole and Hariharan [3] (we will refer to this algorithm as the CH algorithm). It can be easily shown that, after a preprocessing step of $O(m)$ time, the CH algorithm takes $\tilde{O}(nk^3)$ time in each iteration of our while loop. Hence the running time of Algorithm 1.1 is $\tilde{O}(m + n^2k^3)$.

3 An Efficient Algorithm

In this section we present a more efficient algorithm for the at most k-connectivity problem. We show that the work done by the CH algorithm in one iteration of Algorithm 1.1 can be *reused* in another iteration. So, first we need to understand in some detail how the CH algorithm works. This algorithm is basically a constructive proof of the following theorem that appears in [1][2].

THEOREM 3.1. *Given an Eulerian directed graph G and any vertex r, there exist k edge disjoint directionless trees $T_1, T_2, \ldots, T_k$ rooted at r such that each vertex v in G appears exactly $con(v)$ times over all the trees and has in-degree exactly $con(v)$ over these trees, where $con(v)$ is the edge connectivity of r to v and $k = \max_{v \neq r}\{con(v)\}$.*

[2] Actually a slightly stricter version of the theorem appears in the cited reference.

Algorithm 1.1 An algorithm for partial Gomory-Hu tree with parameter k for graph $G = (V, E)$

– Initialize the tree $\mathcal{T}$ to a single node, which is the entire vertex set V.
– Initialize the queue Q to the queue containing only one element, which is the set V.
while the queue Q is not empty **do**
  – delete the first element of Q; call this element S.
  – call the minimum Steiner cut algorithm with the set S as the Steiner set.
  if the value, c, of this minimum Steiner cut is $\leq k$ **then**
    – let S_1 and S_2 be the two components that S is split into by the above cut; update $\mathcal{T}$ by splitting the node S into nodes S_1 and S_2 and introduce an edge of weight c between S_1 and S_2.
    – set the neighbors of S_1 and S_2 in the tree $\mathcal{T}$ appropriately.
    – insert the node S_1 (similarly, S_2) in the queue Q if S_1 (resp., S_2) contains more than one vertex.
  end if
end while

The CH algorithm first uses the Nagamochi-Ibaraki construction [10] as a preprocessing step to reduce the number of edges in the graph to $O(nk)$ where k is the size of the minimum Steiner cut. The undirected graph thus constructed is converted to an Eulerian directed graph by orienting each undirected edge in both directions. Any vertex in the Steiner set S is designated as the root r and the algorithm proceeds by constructing trees rooted at r satisfying the constraints specified in Theorem 3.1. A collection of vertices B becomes a *black supervertex* after constructing i trees if the edge connectivity of r to B is i (i.e., when all the vertices in B are contracted into a single node, the minimum cut separating r from this node is of size i). From this stage onwards, vertices in B are contiguous in all trees they appear in. Note that due to restructuring of the trees, it might so happen that B occurs (possibly multiple times) in some j^{th} tree, where $j > i$ and is absent from a tree $T_j, j \leq i$. The only restriction is that B has to appear exactly i times over all the trees. Also, note that all vertices in B might not appear in all trees, but vertices of B that do appear in a tree have to appear contiguously. This leads to a hierarchical structure of black supervertices.

The algorithm terminates when a black supervertex containing vertices in S is produced. Let us denote this supervertex by B_S. It is easy to see that $(B_S, V \setminus B_S)$ forms a minimum Steiner cut and B_S is the side of this cut not containing the root r. What is interesting to note is that all the vertices of B_S appear contiguously in all trees at the termination of the algorithm. Further, if one now wants to finds the Steiner connectivity of a different set of vertices (obviously, no Steiner vertex can be part of a black supervertex; otherwise, its connectivity from r has already been decided), then we can continue to construct more trees, and the vertices in B_S shall always remain contiguous in these trees. This means that in further stages of the algorithm, B_S can be thought of as a single vertex. Another property is that B_S is a minimal Steiner min-cut, which means that no proper subset of B_S can be a Steiner min-cut. The CH algorithm runs in $O(nk^3 \log n)$ time and the preprocessing step takes $O(m)$ time. Hence, the algorithm has a total time complexity of $O(m + nk^3 \log n)$ or $\tilde{O}(m + nk^3)$.

We would like to use the properties of the CH algorithm, stated above, in our new algorithm. Let us view our partial Gomory-Hu tree algorithm in terms of its *computation tree* $\mathcal{C}$. Each node in $\mathcal{C}$ consists of a call to the CH algorithm with a particular Steiner set. The subproblem corresponding to the root of $\mathcal{C}$ is the computation of a minimum Steiner cut for the Steiner set V. This is nothing but a global min-cut in G. Let $(U_0, V \setminus U_0)$ be this cut. The CH algorithm which computed the cut $(U_0, V \setminus U_0)$ did so by picking some vertex in V as its root and building l rooted trees, where l is the size of this cut. Call this vertex, which was picked as the root, r and assume that $r \in U_0$.

We had associated the entire vertex set V with the root of $\mathcal{C}$ - with the two children of the root, (call them v_1 and v_2), we associate the sets U_0 and $V \setminus U_0$, respectively. The subproblem associated with the node v_1 in $\mathcal{C}$ is the computation of a minimum Steiner cut with Steiner set U_0. To compute this cut, we do not work with the graph G; we compute this minimum Steiner cut in the graph G_1 that is obtained by contracting all the vertices in $V \setminus U_0$ into a single node s_0. The crucial idea now is the following: *in order to compute this cut* (A, B) *where the vertices in* $U_0 \cup \{s_0\}$ *are partitioned into* A *and* B*, we do not have to start from scratch but continue from where we stopped after building the* l *trees which determined the cut* $(U_0, V \setminus U_0)$. In these l trees, all the vertices in $V \setminus U_0$ appeared contiguously since $V \setminus U_0$ is a black supervertex in the trees - so contracting them and regarding them as one node comes for free. So the first l trees which were built,

can be regarded as a part of the process of computing the minimum Steiner cut for the Steiner set U_0.

More generally, at any stage of our algorithm, when we compute the minimum Steiner cut for a Steiner set S, this set S corresponds to a node in the partial Gomory-Hu tree and the subtrees rooted at each neighbor of this node in the partial Gomory-Hu tree are contracted to single nodes or *supervertices*. We compute the minimum Steiner cut for the Steiner set S in this graph. And most importantly, this minimum Steiner cut need not be computed from scratch and the trees used at its parent in $\mathcal{C}$ can be reused if the root node of those trees belongs to S.

In fact, it is not just that the trees used at a node in $\mathcal{C}$ can be reused in one of its child subproblems, but we will show that it is likely that we can reuse the trees in the *larger* of the two child subproblems. We will show that when we build trees for a minimum Steiner cut computation, if we choose the root r of these trees uniformly at random from the Steiner set and if we consider only *minimal* Steiner min-cuts, then it is likely that the root shall lie on the larger of the two sides of this minimal Steiner min-cut. Hence we have the first few trees already constructed for the larger subproblem. This argument will be formally stated and proved in Section 3.1. Our final result is stated below as Theorem 3.2.

THEOREM 3.2. (MAIN THEOREM) *On an input graph with $O(nk)$ edges, the successive calls to the CH algorithm have a cumulative expected time complexity of $O(nk^3 \log^2 n)$, where k is the maximum value of a minimum Steiner cut for any iteration of the algorithm.*

Now we present our improved algorithm as Algorithm 3.1. The algorithm essentially performs a breadth first walk on the computation tree $\mathcal{C}$, starting from the root where the entire vertex set V is the Steiner set and it ignores all computation subtrees whose edge connectivity has already exceeded k.

The algorithm uses a queue Q which stores the list of Steiner sets for which the CH algorithm has to be called. Additionally, it stores the number of trees already constructed and the trees themselves along with each Steiner set. This is because we will reuse trees constructed earlier at a node for one of its child subproblems. For any subproblem extracted from the queue, if no tree has already been constructed for this subproblem, then the algorithm chooses a root uniformly at random from the Steiner set. The algorithm then proceeds to compute trees using the CH technique and stops either if it crosses $k+1$ trees or if the Steiner connectivity has been computed already (which is indicated by the presence of a black supervertex B_j containing some Steiner vertex). In the first case, the Steiner set is discarded since it is at least $(k+1)$-connected. On the other hand, in the second case, the partial Gomory-Hu tree $\mathcal{T}$ is modified accordingly and two new subproblems are spawned on S_1 and S_2, with trees constructed at this stage preserved for reuse in the subproblem for S_2 since $r \in S_2$ (because $S_1 \subseteq B_j$ and $r \notin B_j$).

3.1 Analysis. We shall now prove the Main Theorem (Theorem 3.2), which we stated earlier in Section 3. We shall need the following lemmas.

Lemma 3.1 is straightforward.

LEMMA 3.1. *In our algorithm for the at most k-connectivity problem on an input graph of $O(nk)$ edges, each node of the computation tree $\mathcal{C}$ has a time complexity of $O(nk^3 \log n)$.*

Proof. Recall that we have restricted ourselves to input graphs with $O(nk)$ edges. So, the time complexity of the first call to the CH algorithm is $O(nk^3 \log n)$, according to [3]. In the subsequent phases, shrinking of vertex sets into single nodes leads to a decrease in the number of edges and vertices. Also note that in each node of the computation tree $\mathcal{C}$, the Steiner connectivity is either at most k or we stop after constructing $k+1$ trees. So, each node of $\mathcal{C}$ has a time complexity of $O(nk^3 \log n)$.

By bounding the running time of each subproblem by $O(nk^3 \log n)$, the child subproblem where the trees constructed for the parent can be reused comes completely for free, since the number of trees to be constructed in the child is also upper bounded by k, and we have already accounted for construction of k trees in the parent.

Lemma 3.2 gives the recurrence relation for the running time of a subproblem in terms of its child subproblems.

LEMMA 3.2. *In our algorithm for the at most k-connectivity problem, let $C = (S, \Gamma)$ denote a node in the computation tree $\mathcal{C}$ with Steiner set S and set of supervertices Γ. Let $s = |S|$ and $\gamma = |\Gamma|$. Let the two children of C in $\mathcal{C}$ be denoted by $C_1 = (S_1, \Gamma_1)$ and $C_2 = (S_2, \Gamma_2)$. Let $s_1 = |S_1|$, $s_2 = |S_2|$, $\gamma_1 = |\Gamma_1|$ and $\gamma_2 = |\Gamma_2|$. If $T(s, \gamma)$ denotes the expected cumulative time complexity of all proper descendants[3] of C in $\mathcal{C}$, then*

$$
\begin{aligned}
T(s,\gamma) \;=\; & T(s_1,\gamma_1) + T(s_2,\gamma_2) \\
& + \frac{s_1}{s}(s_2+\gamma_2)k^3 \log(s_2+\gamma_2) \\
& + \frac{s_2}{s}(s_1+\gamma_1)k^3 \log(s_1+\gamma_1).
\end{aligned} \tag{3.1}
$$

[3] In a tree, vertex v is a *proper descendant* of vertex u if either v is a child of u or the parent of v is a proper descendant of u.

Algorithm 3.1 A faster algorithm for a partial Gomory-Hu tree with parameter k for $G = (V, E)$

– Initialize the tree $\mathcal{T}$ to a single node containing the entire vertex set V.
– Initialize the queue Q to the queue containing the single element $(V, 0, \emptyset)$.
{*The format of a tuple in the queue is (Steiner set, number of trees already constructed, these trees).*}
while the queue Q is not empty **do**
  – $flag :=$ TRUE
  – delete the first element $(S, i, \{T_1, \ldots, T_i\})$ from Q.
  – if $i = 0$ then pick a vertex in S uniformly at random as the root r.
  while $i \leq k$ **and** $flag =$ TRUE **do**
    – construct tree T_{i+1} rooted at r.
    – find black supervertices.
    if $\exists$ some black supervertex B_j such that $B_j \cap S \neq \emptyset$ **then**
      – $S_1 := B_j \cap S$;
      – $S_2 := S \setminus S_1$;
      – Split node S into S_1 and S_2 and connect them by an edge of weight i in $\mathcal{T}$.
      – Set the neighbors of S_1 and S_2 in $\mathcal{T}$ appropriately.
      – if $|S_1| > 1$ then insert $(S_1, 0, \emptyset)$ in the queue Q.
      – if $|S_2| > 1$ then insert $(S_2, i, \{T_1, \ldots, T_i\})$ in the queue Q.
      {*The trees used for S are being reused for S_2 since the root r of these trees is in S_2.*}
      – $flag :=$ FALSE
    else
      – $i := i + 1$
    end if
  end while
end while

Proof. The expected time $T(s, \gamma)$ needed for all proper descendants of C is the sum of the expected time needed at C_1, the expected time needed at C_2, and the expected times needed for the proper descendants of C_1 and C_2. The expected times needed for the proper descendants of C_1 and C_2 are $T(s_1, \gamma_1)$ and $T(s_2, \gamma_2)$, respectively. Let us now calculate the expected time needed at C_1 and C_2.

The root of the trees constructed for the node C in the computation tree was chosen uniformly at random from S. Hence the probability that the root is a vertex in S_1 is s_1/s. If the root is in S_1, then the computation at C_1 comes for free and the computation at C_2 takes $(s_2 + \gamma_2)k^3 \log(s_2 + \gamma_2)$ time. Otherwise, the root is in S_2, in which case the computation at C_2 comes for free and the computation at C_1 takes $(s_1+\gamma_1)k^3 \log(s_1+\gamma_1)$ time. Thus the expected time needed at the nodes C_1 and C_2 is $(s_1/s) \cdot (s_2+\gamma_2)k^3 \log(s_2+\gamma_2) + (s_2/s) \cdot (s_1 + \gamma_1)k^3 \log(s_1 + \gamma_1)$. Thus our recurrence for $T(s, \gamma)$ is

$$\begin{aligned} T(s,\gamma) &= T(s_1,\gamma_1) + T(s_2,\gamma_2) \\ &\quad + \frac{s_1}{s}(s_2+\gamma_2)k^3 \log(s_2+\gamma_2) \\ &\quad + \frac{s_2}{s}(s_1+\gamma_1)k^3 \log(s_1+\gamma_1). \end{aligned}$$

However, we need to consider a subtle point here. The equation above is valid if our algorithm always picks the same minimum Steiner cut to split C on, irrespective of the choice of root at C. However, note that the CH algorithm does not return a single Steiner min-cut, but all the Steiner min-cuts that are minimal on the side not containing the root[4]. Our algorithm needs to decide which Steiner min-cut to split C on. Note that we are always interested in retaining as many vertices as possible in the part of the Steiner cut which contains the root since we shall get the corresponding subproblem for free in the next iteration. So, the algorithm always splits C along that particular cut (among the minimal Steiner min-cuts returned by the CH algorithm) which retains the maximum number of vertices on the side of the root. This ensures that the time complexity obtained in the analysis is an upper bound on the actual time complexity, since the root component in the actual algorithm shall be at least as large as S_1 or S_2 (depending on where the root lies).

We bound $\log(s_i + \gamma_i)$ by $\log n$ for $i = 1, 2$ in Equation (3.1) and define $T^*(\cdot, \cdot)$ as $T(\cdot, \cdot) = T^*(\cdot, \cdot)k^3 \log n$.

[4]These are Steiner min-cuts where the side not containing the root has edge connectivity strictly greater than the Steiner connectivity.

Then Equation (3.1) becomes

$$\begin{aligned} T^*(s,\gamma) &= T^*(s_1,\gamma_1)+T^*(s_2,\gamma_2) \\ &\quad +\frac{s_1}{s}(s_2+\gamma_2) \\ &\quad +\frac{s_2}{s}(s_1+\gamma_1). \end{aligned}$$

This simplifies to

$$\begin{aligned} T^*(s,\gamma) &= T^*(s_1,\gamma_1)+T^*(s_2,\gamma_2) \\ &\quad +\frac{2s_1s_2}{s}+\frac{\gamma_1s_2+\gamma_2s_1}{s}. \end{aligned} \tag{3.2}$$

LEMMA 3.3. *The cost due to the term $2s_1s_2/s$ in the expression for $T^*(s,\gamma)$ in Equation (3.2) adds to a total cost of $O(n\log n)$ over all the nodes in the computation tree $\mathcal{C}$.*

Proof. Let us assume, without loss of generality, that $s_1 \le s_2 < s$. Since $2s_1s_2/s \le 2s_1$ (because $s_2 < s$), we will distribute the cost by assigning a charge of 2 to each vertex in S_1. It may be noted that any given vertex in the input graph is an element of the Steiner set in successive nodes along a path of the computation tree $\mathcal{C}$ since the Steiner sets of any two sibling nodes of $\mathcal{C}$ are mutually disjoint. It is this path only that contributes to the charge assigned to a vertex.

Further, note that when a vertex gets a charge of 2 from a node in the computation tree, it is accompanied by a reduction of the size of the Steiner set by at least a factor of 2. This is because vertices in S_1, which are fewer in number than vertices in S_2, (by our assumption that $s_1 \le s_2$) are being charged. Obviously, on any path of the computation tree, the size of the Steiner set can halve at most $\log n$ times, since the number of vertices in the Steiner set associated with the root in $\mathcal{C}$ is n. Since only vertices in a Steiner set are being charged and no vertex gets a cost of more than $2\log n$, the cumulative cost over all the vertices is $O(n\log n)$.

LEMMA 3.4. *The cost due to the term $(\gamma_1s_2+\gamma_2s_1)/s$ in the expression for $T^*(s,\gamma)$ in Equation (3.2) adds to a total cost of $O(n\log n)$ over all the nodes in the computation tree $\mathcal{C}$.*

To prove the above lemma, we shall need Lemma 3.5.

LEMMA 3.5. *If $n_0, n_1, n_2, \ldots$ are positive integers, where $n_0 > n_1$, $n-n_1 > n_2$, $n-n_1-n_2 > n_3, \ldots$ and $n_0 > 2$, then*

$$\frac{n_1}{n_0}+\frac{n_2}{n_0-n_1}+\frac{n_3}{n_0-n_1-n_2}+\ldots < \ln n_0.$$

Proof. We prove this lemma by induction. The base case is easy to see. Since $n_0 > 2$, $\ln n_0 \ge 1 > n_1/n_0$. Inductively assume that

$$\frac{n_2}{n_0-n_1}+\frac{n_3}{n_0-n_1-n_2}+\ldots < \ln(n_0-n_1) \tag{3.3}$$

Now, we know that $\ln(1-x) < -x$, for $0 < x < 1$, since $1-x < e^{-x}$. Setting $x = n_1/n_0$, we have

$$\ln\frac{n_0}{n_0-n_1} > \frac{n_1}{n_0} \tag{3.4}$$

Combining Inequalities (3.3) and (3.4), we get the following inequality, which proves our claim.

$$\frac{n_1}{n_0}+\frac{n_2}{n_0-n_1}+\frac{n_3}{n_0-n_1-n_2}+\ldots < \ln n_0.$$

Proof. (of LEMMA 3.4) The term $(\gamma_1s_2+\gamma_2s_1)/s$ in Equation (3.2) represents the cost in the CH algorithm due to supervertices, i.e., the nodes that are created by contracting subsets of vertices. We charge each supervertex for its own cost. Each supervertex moves down a path in $\mathcal{C}$ and hence receives charge only for computations along a path. Also, note that if a supervertex moves into a child node from its parent node in $\mathcal{C}$, it is charged the ratio of the size of the Steiner set in its sibling to that of the Steiner set in the parent node. Hence, the total charge received by a supervertex in the overall computation is of the form $n_1/n_0 + n_2/(n_0-n_1) + n_3/(n_0-n_1-n_2)+\ldots$, where $n > n_0$, $n_0 > n_1$, $n_0-n_1 > n_2$, $n_0-n_1-n_2 > n_3, \ldots$.

By Lemma 3.5, we can bound the total charge on a supervertex by $\ln n_0$ and therefore, by $\ln n$. We also note that the total number of supervertices increases by exactly 2 from a parent to its children in $\mathcal{C}$. This follows from the fact that the supervertices represent neighbors (actually, subtrees subtended at the neighbors) of a node in the partial Gomory-Hu tree. When a node splits into two, each neighbor of the original node gets assigned to exactly one of the two new nodes spawned. Also, the two new nodes are neighbors. So, the total number of supervertices increases by exactly 2 from a parent node in $\mathcal{C}$ to its children subproblems. Initially there are no supervertices in the root node of $\mathcal{C}$ and each node creates at most 2 supervertices. Since there are at most $n-1$ computation nodes in $\mathcal{C}$, the total number of supervertices is $O(n)$. Hence, the cumulative cost over all the supervertices is $O(n\log n)$.

Proof. (MAIN THEOREM) We note that $T^*(s_1,\gamma_1)$ and $T^*(s_2,\gamma_2)$ are recursive terms in the expression for $T^*(s,\gamma)$ in Equation (3.2) and are therefore accounted for at a different level of the computation tree $\mathcal{C}$. Also, $T^*(1,*) = 0$. Thus, we conclude that $T^*(n,0) = O(n\log n)$. This implies that $T(n,0) = O(nk^3\log^2 n)$.

Recall that we defined $T(s, \gamma)$ on a node by excluding the cost of the call to the CH algorithm at that node itself. So the total time complexity is $T(n, 0)$ and the cost due to the invocation of the CH algorithm at the root node of $\mathcal{C}$. The latter cost is $O(nk^3 \log n)$ by Lemma 3.1. Hence, the total expected time complexity of our algorithm on a graph with at most $O(nk)$ edges is $O(nk^3 \log^2 n)$ or $\tilde{O}(nk^3)$. This proves the Main Theorem.

Now it is easy to show that the expected running time of our at most k-connectivity algorithm is $\tilde{O}(m + nk^3)$ for any input graph on m edges. We use the Nagamochi-Ibaraki construction [10] as our preprocessing step. In time $O(m)$, this makes the number of edges in the input graph $O(nk)$, without changing the size of any cut smaller than $k + 1$ and also not creating any new cut of size at most k. Now, our at most k-connectivity algorithm on this graph takes expected $\tilde{O}(nk^3)$ time according to the Main Theorem. Hence, the total expected running time is $\tilde{O}(m + nk^3)$. This completes the proof of Theorem 1.1, which was stated in Section 1.

3.2 The approximate at most αc-connectivity problem. Clearly, our algorithm solves the at most αc-connectivity problem in $\tilde{O}(m + nc^3)$ time, where c is the value of min-cut of G and $\alpha \geq 1$ is some constant. We now present an approximation algorithm for this problem. Our algorithm is based on uniform random sampling of the edges in the input graph. We state the following theorem, due to Karger, that appears in [9].

THEOREM 3.3. *Let c be the size of the global minimum cut in an undirected graph G. Build G^* by including each edge from G with probability p. If $p > 8 \log n/\epsilon^2 c$ then with probability $1 - 1/n$ every cut in G^* has value within $(1 \pm \epsilon)$ of its expectation.*

We use this theorem to give an approximation algorithm for the at most αc-connectivity problem. Apply uniform sampling over all edges in the input graph G with sampling probability $p = 32 \log n/\epsilon^2 c$, where ϵ is our error parameter, to produce a skeleton graph G^*. The expected size of the global minimum cut in G^* is $O(\log n/\epsilon^2)$.

Run Algorithm 3.1 on G^* to compute all pairs of vertices whose edge connectivity is at most $(1+\epsilon/2)\alpha pc$, which is $O(\alpha \log n/\epsilon^2)$. The advantage we get out of sampling is that the time complexity of the CH algorithm reduces substantially since the size of the minimum Steiner cuts are scaled down roughly by a factor of c (and scaled up by a factor of $\log n$). The Uniform Sampling Theorem for Cuts guarantees that the edge connectivity between any two vertices returned by the algorithm is within $(1 \pm \epsilon/2)$ of its expected value with a probability of at least $1 - 1/n$. We have thus shown Theorem 1.2. This gives us an almost linear time Monte Carlo algorithm for this problem, when α and ϵ are constants.

All-pairs approximate min-cuts. Another useful corollary of our result is in computing all-pairs approximate edge connectivities in undirected unweighted graphs. Since the current best algorithm for all-pairs *exact* min-cuts could take $\tilde{\Theta}(n^{11/3})$ time [6, 8] (through $n - 1$ max-flow computations using the Goldberg-Rao max-flow algorithm [5]), a useful way of estimating the edge connectivity of each pair of vertices is to compute all-pairs *approximate* min-cuts. Using a sampling scheme of Benczúr and Karger [2], we can compute $(1 \pm \epsilon)$ estimates of every s-t minimum cut in $\tilde{O}(n^{5/2}/\epsilon^2)$ time. So when ϵ is a constant, computing all-pairs approximate min-cuts takes $\tilde{O}(n^{5/2})$ time.

We can improve this result by additionally computing all s-t edge connectivities which are at most $\sqrt{n}$ using our at most k-connectivity algorithm with $k = \sqrt{n}$. Our algorithm takes expected $\tilde{O}(n^{5/2})$ time. Once all s-t edge connectivities of at most $\sqrt{n}$ have been computed *exactly*, we use the sampling scheme of Benczúr and Karger [2] to construct a weighted sparse skeleton graph, on which we run the max-flow computations to *approximately* compute the edge connectivities of the remaining pairs of vertices. This takes $\tilde{O}(n^{5/2})$ time as well. Thus in expected $\tilde{O}(n^{5/2})$ time, we compute all-pairs approximate min-cuts with the additional guarantee that we get the *exact* value of each minimum s-t cut whose value is at most $\sqrt{n}$. Such a guarantee would be valuable since those minimum s-t cuts whose value is low correspond to fragile parts of the network, and it would be important to know exactly what their edge connectivity is, instead of just an approximate estimate.

4 Minimum $\mathbb{T}$-cut problem

Given a set of vertices $\mathbb{T} \subseteq V$ of even cardinality, a cut $(A, V \setminus A)$ such that $|A \cap \mathbb{T}|$ (and hence, $|(V \setminus A) \cap \mathbb{T}|$) is odd is called a $\mathbb{T}$-cut. The minimum $\mathbb{T}$-cut problem asks for a $\mathbb{T}$-cut of minimum size. For solving this problem, we use the following version of submodularity from [12].

THEOREM 4.1. *If $\mathbb{T}_1$ and $\mathbb{T}_2$ are even cardinality subsets of V, and $(S_1, V \setminus S_1)$ is a minimum $\mathbb{T}_1$-cut but $|S_1 \cap \mathbb{T}_2|$ is even, then there exists a minimum $\mathbb{T}_2$-cut $(S_2, V \setminus S_2)$ such that $S_2 \subset S_1$ (or $S_1 \subset S_2$).*

This theorem implies the following lemma.

LEMMA 4.1. *Let $(S, V \setminus S)$ be a minimum Steiner cut for Steiner set $\mathbb{T}$. Then, either*

Algorithm 4.1 $minTcut(G, \mathbb{T})$: Minimum $\mathbb{T}$-cut in the graph $G = (V, E)$

– Find the minimum Steiner cut $(S, V \setminus S)$ with Steiner set $\mathbb{T}$.
if $|S \cap \mathbb{T}|$ is *odd* **then**
– return $(S, V \setminus S)$.
else
– $\mathbb{T}_1 = S \cap \mathbb{T}$
– $\mathbb{T}_2 = \mathbb{T} \setminus \mathbb{T}_1$
– Construct G_1 and G_2 by contracting $V \setminus S$ and S respectively.
– **return** $\min(minTcut(G_1, \mathbb{T}_1), minTcut(G_2, \mathbb{T}_2))$.
end if

(a) $S \cap \mathbb{T}$ is of odd cardinality, in which case $(S, V \setminus S)$ is a minimum $\mathbb{T}$-cut, or,
(b) there exists a minimum $\mathbb{T}$-cut $(S^*, V \setminus S^*)$ such that $S^* \subset S$ (or $S \subset S^*$).

Proof. All $\mathbb{T}$-cuts being Steiner cuts as well, a minimum Steiner cut is also a minimum $\mathbb{T}$-cut if it splits $\mathbb{T}$ into odd fragments. This proves part (a) of the statement. Suppose the minimum Steiner cut is not a $\mathbb{T}$-cut. Then choose vertices u and v such that $u \in S \cap \mathbb{T}$ and $v \in (V \setminus S) \cap \mathbb{T}$. Obviously, $(S, V \setminus S)$ is a minimum u-v cut. Now, using Theorem 4.1, we can claim that there exists a minimum $\mathbb{T}$-cut $(S^*, V \setminus S^*)$ such that $S^* \subset S$ (or $S \subset S^*$).

Lemma 4.1 essentially states that a minimum Steiner cut for Steiner set $\mathbb{T}$ (call this cut (A, B)) is either a minimum $\mathbb{T}$-cut or the problem reduces to computing the smaller of these two cuts: (i) a min $(A \cap \mathbb{T})$-cut in the graph where B is contracted to a single node, (ii) a min $(B \cap \mathbb{T})$-cut in the graph where A is contracted to a single node. We give this algorithm as Algorithm 4.1.

We make our algorithm faster by using the Nagamochi-Ibaraki (NI) algorithm [10] as a preprocessing step. This algorithm constructs a series of spanning forests (call them NI spanning forests) using edges in the graph such that the edges in the first i spanning forests, for any i, form a subgraph that satisfies the following properties:
(i) Any cut having cardinality at most i in the original graph retains all its edges in the subgraph.
(ii) Any cut having cardinality greater than i in the original graph has cardinality at least i in the subgraph.

Obviously, the first i NI spanning forests have at most $(n-1)i$ edges. So, this algorithm gives a sparse skeleton graph which retains the edge connectivity properties of the original graph up to a threshold value.

All the NI spanning forests can be constructed in $O(m)$ time in an unweighted undirected graph [10]. After constructing these forests, our algorithm proceeds in several iterations. The critical observation is that we need to consider edges in the first i NI spanning forests only for identifying the Steiner cuts of $\mathbb{T}$ having cardinality at most $i-1$. So, we start off with the first NI spanning forest and try to construct one directionless spanning tree using the CH algorithm on these edges. If we are able to construct the tree, we add the edges in the second NI forest and try to construct the second directionless tree, and so on. If we are unable to construct the first tree, we obtain a Steiner cut of size 0. We are guaranteed by the property of the NI forests mentioned above that this cut indeed has 0 edges in the original graph. So, we have obtained a genuine minimum Steiner cut. If this cut is a $\mathbb{T}$-cut, we have found a minimum $\mathbb{T}$-cut as well; otherwise, we split the vertices along this cut to spawn two new subproblems and try to construct one tree in each of the new subproblems, and so on.

In general, after the $(i-1)$th iteration, we are left with a set of subproblems, where each Steiner set is a subset of $\mathbb{T}$ having edge connectivity at least $i-1$. Also, for each of these subproblems, $i-1$ directionless trees have already been constructed using the edges in the first $i-1$ NI spanning forests only. For a particular subproblem, if we find a Steiner cut of cardinality $i-1$ after including the edges of the i-th NI spanning forest, either this cut is a $\mathbb{T}$-cut (in which case we output this cut as a minimum $\mathbb{T}$-cut), or we split the Steiner set to spawn two new subproblems and construct $i-1$ directionless trees for these subproblems. If no cut of cardinality $i-1$ is found, then we construct the i-th directionless tree and wait for the next iteration.

We also reuse trees from a parent computation node in one of its children if the root of the trees constructed at the parent node is in the Steiner set of the child, exactly along the lines of Algorithm 3.1 described in Section 3. We present this algorithm as Algorithm 4.2.

The variable *ind* stores the current iteration index and the CH algorithm is constrained to use only the edges in E_{cur}. The algorithm uses two queues - Q_{cur} stores the subproblems in the current iteration and Q_{next} stores the subproblems in the next iteration. The two flags $flag1$ and $flag2$ are used to indicate that the minimum $\mathbb{T}$-cut has not been found, and that the minimum Steiner cut has not been found for the current subproblem, respectively.

The analysis of the algorithm is identical to that of Algorithm 3.1, except that the length of a path in the computation tree where the size of the Steiner set reduces by half in each step is $\log|\mathbb{T}|$. This leads to a total time complexity of $O(m + nk^3 \log n \log|\mathbb{T}|)$ or

Algorithm 4.2 Our algorithm for minimum $\mathbb{T}$-cut in the graph $G = (V, E)$

– run NI algorithm on G to construct all the NI spanning forests F_1, F_2, ...
– initialize ind to 1.
– initialize the set of edges E_{cur} to F_1.
– initialize the queue Q_{cur} to the queue containing the single element $(\mathbb{T}, 0, \emptyset)$.
– initialize the queue Q_{next} to an empty queue.
– $flag1 :=$ TRUE
while $flag1 =$ TRUE **do**
  while Q_{cur} is not empty **and** $flag1 =$ TRUE **do**
    – delete the first element $(S, i, \{T_1, \ldots, T_i\})$ from Q_{cur}.
    – if $i = 0$ then pick a vertex in S uniformly at random as the root r.
    – $flag2 :=$ TRUE
    while $i < ind$ **and** $flag2 =$ TRUE **do**
      – construct tree T_{i+1} rooted at r using edges in E_{cur} only.
      – find black supervertices.
      if $\exists$ some black supervertex B_j such that $B_j \cap S \neq \emptyset$ **then**
        – $S_1 := B_j \cap S$
        – $S_2 := S \setminus S_1$
        – $flag2 :=$ FALSE
        if $|S_1|$ is *odd* **then**
          – $minTcut := (B_j, V \setminus B_j)$
          – $flag1 :=$ FALSE
        else
          – insert $(S_1, 0, \emptyset)$ in the queue Q_{cur}.
          – insert $(S_2, i, \{T_1, \ldots, T_i\})$ in the queue Q_{cur}.
        end if
      end if
      – $i := i + 1$
    end while
    if $flag2 =$ TRUE **then**
      – insert $(S, ind, T_1, \ldots, T_{ind})$ in the queue Q_{next}.
    end if
  end while
  – $ind := ind + 1$
  – $E_{cur} = E_{cur} \cup F_{ind}$
  – copy Q_{next} into Q_{cur} and flush Q_{next}.
end while
– return $minTcut$.

$\tilde{O}(m+nk^3)$, thus proving Theorem 1.3, which was stated in Section 1.

Conclusions. We presented a fast algorithm for constructing a partial Gomory-Hu tree and used it to solve the at most k-connectivity problem and the minimum $\mathbb{T}$-cut problem. An open problem is to obtain a fast (possibly, approximate) algorithm to construct the *entire* Gomory-Hu tree (rather than a partial Gomory-Hu tree) using our approach.

References

[1] J. Bang-Jensen, A. Frank and B. Jackson, *Preserving and increasing local edge connectivity in mixed graphs*, SIAM J. Discrete Mathematics 8(2) (1995), pp. 155–178.

[2] A. Benczúr and D. R. Karger, *Approximating s-t Minimum Cuts in $\tilde{O}(n^2)$ Time*, J. Algorithms 37(1) (2000), pp. 2–36.

[3] R. Cole and R. Hariharan, *A fast algorithm for computing steiner edge connectivity*, Proc. of the 35th Annual ACM Symposium on Theory of Computing, San Diego (2003), pp. 167–176.

[4] Harold N. Gabow, *A matroid approach to finding edge connectivity and packing arborescences*, J. Comput. System Sci. 50, (1995), pp. 259–273.

[5] A. V. Goldberg and S. Rao, *Beyond the flow decomposition barrier*, JACM 45(5) (1998), pp. 783–797.

[6] R. E. Gomory and T. C. Hu, *Multi-terminal network flows*, J. Soc. Indust. Appl. Math. 9(4) (1961), pp. 551–570.

[7] M. Grötschel, L. Lovász and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, 1988.

[8] D. Gusfield, *Very simple methods for all pairs network flow analysis*, SIAM J. Computing 19(1) (1990), pp. 143–155.

[9] D. R. Karger, *Random Sampling in cut, flow, and network design problems*, Proc. of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, Montréal, Québec, Canada (1994), pp. 648–657.

[10] Hiroshi Nagamochi and Toshihide Ibaraki, *A Linear-Time Algorithm for Finding a Sparse k-Connected Spanning Subgraph of a k-Connected Graph*, Algorithmica 7(5&6) (1992), pp. 583–596.

[11] Hiroshi Nagamochi and Toshimasa Watanabe, *Computing k-Edge-Connected Components of a Multigraph*, Inst. Electron. Inform.Comm, Vol E76-A, 4 (1993), pp. 513-517.

[12] Romeo Rizzi, *A simple minimum T-cut algorithm*, Discrete Applied Mathematics 129(2-3) (2003), pp. 539–544.