On Element-Connectivity Preserving Graph Simplification

Chandra Chekuri[1], Thapanapong Rukkanchanunt[2], and Chao Xu[1]

[1] Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, Urbana, IL, USA
{chekuri,chaoxu3}@illinois.edu
[2] Chiang Mai University, Chiang Mai, Thailand thapanapong.r@cmu.ac.th

Abstract. The notion of *element-connectivity* has found several important applications in network design and routing problems. We focus on a reduction step that preserves the element-connectivity [18,4,3], which when applied repeatedly allows one to reduce the original graph to a simpler one. This pre-processing step is a crucial ingredient in several applications. In this paper we revisit this reduction step and provide a new proof via the use of setpairs. Our main contribution is algorithmic results for several basic problems on element-connectivity including the problem of achieving the aforementioned graph simplification. We utilize the underlying submodularity properties of element-connectivity to derive faster algorithms.

Keywords: element-connectivity, Gomory-Hu tree, reduction, bisubmodular

1 Introduction

Let $G = (V, E)$ be an undirected simple graph. The edge-connectivity between two distinct nodes u, v, denoted by $\lambda_G(u, v)$, is the maximum number of edge-disjoint paths between u and v in G. The vertex-connectivity between u and v, denoted by $\kappa_G(u, v)$ is the maximum number of internally vertex-disjoint paths between u and v. These two connectivity measures are classical and extensively studied in graph theory, combinatorial optimization and algorithms. The notion of *element-connectivity* is more recent and is defined as follows. Let $T \subseteq V$ be a set of terminals; vertices in $V \setminus T$ are referred to as non-terminals. For any two distinct terminals $u, v \in T$, element-connectivity between u and v is the maximum number of u-v paths in G that are pairwise "element"-disjoint where elements consist of edges and non-terminals. Note that element-disjoint paths need not be disjoint in terminals. We use $\kappa'_G(u, v)$ to denote the element-connectivity between u and v. Via Menger's theorem one can characterize edge, vertex and element connectivity in an equivalent way via cuts. We use $\kappa'_G(T) = \min_{u,v \in T} \kappa'_G(u, v)$ to denote the global element-connectivity, the minimum number of elements whose deletion separates some pair of terminals. See Fig. 1 for example.

Element-connectivity can be seen to generalize edge-connectivity by letting $T = V$. At the same time, element-connectivity is also closely related to vertex-connectivity. If T is an independent set then $\kappa'_G(u, v)$ is the maximum number of paths from u to v that are disjoint in non-terminals. In particular, if T contains exactly two vertices s and t, then $\kappa_G(s, t) = \kappa'_G(s, t)$. Element-connectivity has found several applications

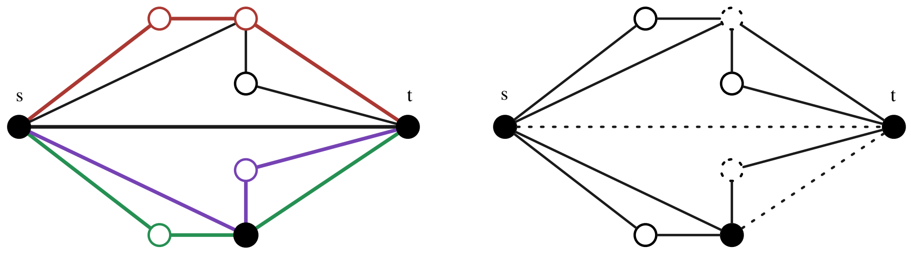

Fig. 1. The black vertices are the terminals. The left image shows 4 element-disjoint st-paths. The right image shows removing 4 elements disconnects s and t. $\kappa'(s,t) = 4$

in network design, routing and related problems, some of which we will discuss later. Several of these applications rely on an interesting graph reduction operation shown first by Hind and Oellermann [18]. To describe their result we use the notation G/pq to denote the graph obtained from G by contracting the edge pq, and $G - pq$ to denote the graph with edge pq deleted.

Theorem 1 (Hind & Oellermann [18]). *Let $G = (V, E)$ be an undirected graph and $T \subseteq V$ be a terminal-set such that $\kappa'_G(T) \geq k$. Let pq be* any *edge where $p, q \in V \setminus T$. Then $\kappa'_{G_1}(T) \geq k$ or $\kappa'_{G_2}(T) \geq k$ where $G_1 = G - pq$ and $G_2 = G/pq$.*

Chekuri and Korula generalized the theorem to show that the same reduction operation also preserves the *local* element-connectivity of every pair of terminals.

Theorem 2 (Chekuri & Korula [3]). *Let $G = (V, E)$ be an undirected graph and $T \subseteq V$ be a terminal-set. Let pq be* any *edge where $p, q \in V \setminus T$ and let $G_1 = G - pq$ and $G_2 = G/pq$. Then one of the following holds: (i) $\forall u, v \in T$, $\kappa'_{G_1}(u, v) = \kappa'_G(u, v)$ (ii) $\forall u, v \in T$, $\kappa'_{G_2}(u, v) = \kappa'_G(u, v)$.*

We refer to the preceding theorem as the reduction lemma following the usage from [3]. By repeatedly applying the reduction lemma, as observed in prior work, we obtain the following corollary.

Corollary 1. *Given a graph $G = (V, E)$ and a terminal set $T \subseteq V$ there is a minor $H = (V', E')$ of G such that (i) $T \subseteq V'$ and (ii) $V' \setminus T$ is an independent set in H and (iii) $\kappa'_H(u, v) = \kappa'_G(u, v)$ for all $u, v \in T$. In particular, if T is an independent set in G then H is a bipartite graph with bipartition $(T, V' \setminus T)$.*

The minor H in the previous corollary is called a *reduced graph* of G. A graph is *reduced* if there are no edges between non-terminals.

Remark 1. A reduced graph $G = (V, E)$ where the terminals T form an independent set can be interpreted as a hypergraph $H = (T, E')$. H contains an edge e_v for every non-terminal v in G, where e_v is the set of neighbors of v in G. Element-connectivity of terminals T in G is equivalent to hypergraph edge-connectivity in H. st-edge connectivity on hypergraphs are defined the same way as graphs : the maximum number of edge disjoint st-paths.

	Min Cut	Min Cut(WHP)	All-pair	All-pair (WHP)	Reduce
λ	$\tilde{O}(m)$[21]	$\tilde{O}(m)$[21]	$\tilde{O}(n^{27/8})$[7]	$\tilde{O}(nm)$[16]	-
κ'	$O(\lvert T\rvert\ \mathrm{MF}(n,m))$	same as all-pair	$O(\lvert T\rvert\ \mathrm{MF}(n,m))$	$\tilde{O}(\lvert T\rvert n^{\omega})$[14],$O(m^{\omega})$[6]	$O(\lvert T\rvert nm)$
κ	$O(n^{7/4}m)$[12]	$\tilde{O}(nm)$[17]	$O(n^{9/2})$[8]	$\tilde{O}(n^{2+\omega})$[14]	-

Fig. 2. The running time for various algorithms for a graph with n vertices and m edges and terminal vertices $|T|$. The row for κ' is our result. MF(n, m) is the running time for a maximum flow on unit capacity directed graph with n vertices and m edges, which is known to be $O(\sqrt{n}m)$[8]. WHP indicates with high probability bounds for randomized algorithms. ω is the matrix multiplication constant. $\tilde{O}$ notation suppresses poly-logarithmic factors.

Our Results: In this paper we revisit Theorem 2 and Corollary 1 from two related view points. The proofs of Theorems 1 and 2 use elementary arguments on flows and cuts in graphs. We obtain an alternative proof of Theorem 2 using submodularity and supermodularity property of setpairs. Although these properties have been used routinely in network design papers that deal with element-connectivity, they have not been applied in the context of the reduction lemma itself. Second, we examine algorithmic aspects of element-connectivity, and this is our main contribution. Very basic questions have not been explored in contrast to substantial literature on edge and vertex connectivity. For instance, how fast can $\kappa'_G(T)$ be computed? How fast can the graph H promised by Corollary 1 be computed? We obtain several results which are summarized in Fig. 2. In particular, given a graph on n nodes with m edges we obtain an algorithm that outputs the reduced graph for a given set of terminals T in $O(|T|nm)$ time. The key observation that underlies the algorithms is that a symmetric submodular function can be defined over the terminal set T that corresponds to the element-connectivity between then. This in turn allows us to compute and exploit a Gomory-Hu tree for this function.

Applications: Element-connectivity has found important applications in three areas: network design, packing element-disjoint Steiner trees and forests, and more recently in routing for node-disjoint paths and related applications (we omit detailed references due to lack of space). Our algorithmic improvements most directly affect the second application, namely the problem of packing element-disjoint Steiner trees and forests. We briefly describe the simpler case of packing element-disjoint Steiner trees which was the original motivation for the graph reduction step [18]. Here we are given a graph $G = (V, E)$ and terminal set T and the goal is to find the maximum number of Steiner trees for T that are pairwise element-disjoint. It is known that in general graphs one can find $\Omega(k/\log |T|)$ trees where $k = \kappa'_G(T)$ and there are examples where this bound is tight [4]. In planar graphs one can find $\Omega(k)$ trees [1,3]. Algorithms for these first need to compute k, and then work with the reduced graph. Computing the reduced graph is the bottleneck and our result thus implies an $O(|T|nm)$-time algorithm for these packing problems; the previous bounds is $O(k|T|^2m^2)$.

Discussion: There is a vast amount of literature on algorithms for computing edge and vertex connectivity in graphs, and related problems on flows and cuts. As the table in Fig 2 shows, the edge connectivity versions have faster algorithms and are much better understood. This is not surprising since edge-connectivity has additional structure that can be exploited, including the existence of a Gomory-Hu tree. In contrast, vertex-connectivity does not admit even a weaker notion of flow-trees [2]. Element-connectivity is an interesting notion that has helped bridge edge and vertex connectivity in various applications, and we believe that studying its computational aspects in more depth is an fruitful direction of research. Our work here can be seen as a first step in exploiting the basic properties of element-connectivity to obtain faster algorithms. In this context we mention the the splitting-off operation to preserve edge-connectivity introduced by Lovász [24] and strengthened by Mader [25]. The algorithmic aspects of splitting-off have been exploited in several papers on edge-connectivity including some recent ones [16,23] and we hope similar ideas may bear fruit for element-connectivity.

2 Preliminaries

A set function $f : 2^V \to \mathbb{R}$ is submodular if for all $A, B \subseteq V, f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$. f is non-negative if $f(A) \geq 0$ for all $A \subset V$. f is symmetric if $f(A) = f(V - A)$ for all $A \subset V$. For a non-negative symmetric submodular function $f : 2^V \to \mathbb{R}$, we define the *f-connectivity* between $s, t \in V$ as:

$$\alpha_f(s,t) = \min_{U \subset V, |U \cap \{s,t\}|=1} f(U). \tag{1}$$

Note that $\alpha_f(s,t) = \alpha_f(t,s)$ for symmetric f. A capacitated spanning tree (R, c) on V is called a *Gomory-Hu tree* (*cut tree*) of f if for all $st \in E(R)$, $f(A) = \alpha_f(s,t) = c(st)$, where A is a component of $R - st$. $\alpha_f(s,t)$ for all $s, t \in V$ can be read off from the Gomory-Hu tree as the smallest capacity on the unique path between s and t. A Gomory-Hu tree always exists when f is non-negative, symmetric, and submodular (see [26]).

Given an undirected graph $G = (V, E)$, for any $W \subseteq V$, we denote by $\delta_G(W)$ the set of edges with exactly one end point in W. The cut function f, defined by $f(A) = |\delta_G(A)|$, is a symmetric submodular function. For $A, B \subseteq V$, $E(A, B)$ denotes the set of edges with one end point in A and the other in B.

Set pairs, bisubmodularity and bisupermodularity: Given a finite ground set V, a *setpair* $W = (W_t, W_h)$ is an ordered pair of disjoint subsets of V; W_t or W_h may be empty. Let $\mathcal{S}$ be the set of all setpairs of V and denote the set of edges with one endpoint in W_t and the other in W_h by $\delta(W) = E(W_t, W_h)$. Given two setpairs $W = (W_t, W_h)$ and $Y = (Y_t, Y_h)$, let $W \otimes Y$ denote the setpair $(W_t \cup Y_t, W_h \cap Y_h)$, and let $W \oplus Y$ denote the setpair $(W_t \cap Y_t, W_h \cup Y_h)$. Additionally, we define $\overline{(W_t, W_h)} = (W_h, W_t)$. Note that $\delta(W) = \delta(\overline{W})$ for an undirected graph and $\overline{W \otimes Y} = \overline{W} \oplus \overline{Y}$.

A function $f : \mathcal{S} \to \mathbb{R}$ is called *bisubmodular* if $f(W) + f(Y) \geq f(W \otimes Y) + f(W \oplus Y)$ for any two setpairs W and Y. For *bisupermodular* and *bimodular* functions, we change $\geq$ to $\leq$ and $=$ respectively. f is *symmetric* if $f(W) = f(\overline{W})$ for all $W \in \mathcal{S}$.

Definition 1. *Given a ground set V, a non-negative function g on $\mathcal{S}$ is called skew-bisupermodular if for any two setpairs W and Y, one of the following holds:*

$$g(W) + g(Y) \leq g(W \otimes Y) + g(W \oplus Y) \tag{2}$$

$$g(W) + g(Y) \leq g(\overline{W} \otimes Y) + g(\overline{W} \oplus Y) \tag{3}$$

3 Element-connectivity and connections to submodularity

It is natural to work with cut-functions that capture element-connectivity. We define a cut function $C_G : 2^T \to \mathbb{R}_+$ over the terminals as follows: for $U \subset T$, $C_G(U)$ is the minimum number of elements whose removal disconnects U from $T \setminus U$ in G. To formally define $C_G(U)$ we consider vertex tri-partitions (A, Z, B) where $U \subseteq A$ and $(T \setminus U) \subseteq B$; among all such tri-partitions the one with minimum $|Z| + |E(A, B)|$ defines $C_G(U)$. $Z \cup E(A, B)$ is called a *cut-set*.

Theorem 3 (Menger's theorem for element-connectivity). *For a graph G with terminal vertices T, for all $s, t \in T$, $\kappa'_G(s, t) = \min\{C_G(U) : U \subset T, |U \cap \{s, t\}| = 1\}$.*

If T is a independent set in the previous theorem, then the cut-set can be taken to contain no edges.

For our purposes a crucial observation is that C_G is a non-negative symmetric submodular function.

Theorem 4. *Let G be a graph with terminal vertices T. C_G is a non-negative symmetric submodular function over T.*

The preceding theorem implies that C_G admits a Gomory-Hu tree.

4 Algorithmic aspects of element-connectivity

In this section we describe our algorithmic contributions to element-connectivity. In particular we describe how the running times in the second row of the table in Fig. 2 can be realized. Our main contribution is a faster algorithm for graph reduction. In the entire section, we are always working with a graph $G = (V, E)$ with n vertices, m edges and terminal vertices T.

Equivalent directed graph: One view of element-connectivity that greatly helps with computation is to define a flow problem. One can see that $\kappa'_G(s, t)$ is the maximum s-t-flow in G with unit capacities on the edges *and* non-terminal vertices (terminal vertices have no capacity constraint). This prompts us to define a equivalent directed graph, which we get from applying the standard vertex split operation for a graph when there are vertex capacities.

Let $N = V \setminus T$ be the set of non-terminals. Let $N^- = \{v^- | v \in N\}$ and $N^+ = \{v^+ | v \in N\}$. The *equivalent directed graph* of G, denoted by $\tilde{G} = (\tilde{V}, \tilde{E})$, where $\tilde{V} = N^- \cup N^+ \cup T$ and the arc set $\tilde{E}$ is obtained from G as follows:

1. For every $v \in N$, $(v^-, v^+) \in \tilde{E}$.
2. For every $uv \in E$ where $u, v \in N$, $(u^+, v^-), (v^+, u^-) \in \tilde{E}$.
3. For every $uv \in E$ where $u \in T, v \in N$, $(u, v^-), (v^+, u) \in \tilde{E}$.
4. For every $uv \in E$ where $u, v \in T$, $(u, v), (v, u) \in \tilde{E}$.

All the arcs in $\tilde{G}$ implicitly have unit capacity. Any maximum integral acyclic s-t-flow f_{st} in $\tilde{G}$ corresponds to a set of maximum element-disjoint s-t-paths in G. Hence we do not distinguish between maximum flows in $\tilde{G}$ and maximum element-disjoint paths in G. A flow in $\tilde{G}$ contains a vertex v (edge e) in G to mean the corresponding element-disjoint path contains vertex v (edge e).

Lemma 1. $\lambda_{\tilde{G}}(s, t) = \kappa'_G(s, t)$ *for all $s, t \in T$.*

4.1 Computing element-connectivity

Single pair element-connectivity The equivalent directed graph allows us to compute local element-connectivity by running a single maximum flow on a unit capacity directed graph.

Lemma 2. *$\kappa'_G(s,t)$ can be computed in $O(\mathrm{MF}(n,m))$ time.*

Note that if $T = \{s,t\}$ then $\kappa'_G(s,t) = \kappa_G(s,t)$ and moreover maximum bipartite matching can be reduced to $\kappa_G(s,t)$. Thus, improving the time to compute $\kappa'_G(s,t)$ is not feasible without corresponding improvements to other classical problems.

All-pair element connectivity To compute $\kappa'_G(s,t)$ for all pairs $s,t \in T$, we can compute the Gomory-Hu tree that we know exists from Theorem 4. Unlike the single-pair case where element-connectivity behaves like vertex-connectivity, in the all-pair case it is closer to edge-connectivity, and in particular there are at most $|T|-1$ distinct element-connectivity values. A Gomory-Hu tree (R,c) representing the all-pair element-connectivities can be built recursively by solving $|T|-1$ minimum element cut computations, which correspond to maximum flow computations in the equivalent directed graph. Hence all-pair element-connectivity takes $|T|-1$ maximum flows. For dense graphs, if we allow randomization, maximum flow can be solved in $\tilde{O}(n^\omega)$ time[14], where ω is the matrix multiplication constant.

There is an alternative approach for sparse graphs using network coding. Cheung *et al.* describe a randomized algorithm that computes the edge-connectivity in a *directed* graph between *every* pair of vertices in $O(m^\omega)$ time with high probability [6]. Since $\kappa'_G(s,t) = \lambda_{\tilde{G}}(s,t)$ for all $s,t \in T$, all-pair element-connectivity can also be computed in $O(m^\omega)$ time with high probability.

Global element connectivity The global element-connectivity $\kappa'_G(T)$ can be easily obtained from the all-pair problem.

Theorem 5. *$\kappa'_G(T)$ can be computed in $O(|T|\ \mathrm{MF}(n,m))$ time.*

A different algorithm that results $|T|-1$ maximum flows computations can be obtained via an approach similar to that of Hao-Orlin's algorithm [15]. We defer the details to full version of the paper. It is a very interesting open problem to improve the running time for computing $\kappa'_G(T)$. Global edge-connectivity admits a near-linear time algorithm [20,21], but global vertex connectivity algorithms are much slower.

Remark 2. For the special case when G is already a reduced graph, an $O(mn)$ time algorithm exists via the problem of computing a minimum cut in a hypergraph [22].

4.2 Computing a reduced graph

This section highlights the main result of the paper, an $O(|T|nm)$ time algorithm to find a reduced graph. For G a graph with terminals T, H is called a *reduction* of G if it can be obtained from G by a sequence of reduction operations.

The reduction lemma suggests a simple algorithm: pick any edge incident to two nonterminal vertices, check which one of the two operations preserves element-connectivity, reduce and repeat. For a graph on n vertices and m edges, compute all-pair element-connectivity from scratch. The naive scheme when combined with the non-obvious $O(|T|\ \text{MF}(n,m))$ algorithm for all-pair case would take $O(|T|m\ \text{MF}(n,m))$ time. Cheriyan and Salavatipour [4] described exactly this algorithm where they in fact computed all-pair connectivity using $|T|^2$ max-flow computations; their focus was not in improving the running time. We obtain the following

Theorem 6. *For a graph G with n vertices, m edges and terminals T, a reduced graph of G can be computed in $O(|T|nm)$ time.*

The two high-level ideas to obtain an improved run-time are the following.

1. The algorithm initially computes and then maintains a Gomory-Hu tree (R, c) for the element-connectivity of T. For each edge $st \in E(R)$ it maintains a corresponding maximum flow between s and t in $\tilde{G}$ as it evolves with reduction operations.
2. Instead of considering reduction operations on an edge by edge basis in an ad-hoc manner we consider all edges incident to a non-terminal vertex v and process them as a batch.

The first idea alone would give us a run-time of $O(|T|m^2)$. The second idea gives a further improvement to $O(|T|nm)$.

Reduction by vertex elimination: We call an edge pq between two non-terminals a *reducible* edge. For a given non-terminal v let $D(v)$ be the set of all reducible edges incident to v. We say v is *active* if $D(v) \neq \emptyset$. An *elimination* operation on an active vertex v either contracts an edge in $D(v)$ or removes *all* edges in $D(v)$. If the graph is not reduced, there is always an elimination operation that preserves element-connectivity. Indeed, if $D(v)$ cannot be removed, then consider the edges in $D(v)$ in an arbitrary but fixed sequence and apply the reduction operation. At some point there is an edge e such that removing it reduces the element-connectivity. By reduction lemma, we can contract e. An elimination reduces the number of active vertices by at least 1. There can only be $O(n)$ eliminations. Moreover, crucially, we can implement a vertex elimination operation in the same amount of time as an edge reduction operation.

Our goal is to decide quickly whether an active vertex v can be eliminated (that is, $D(v)$ can be removed) and if not which of the edges in $D(v)$ can be contracted. For this purpose we define a weighting of edges of E as follows. First we order the edges in $D(v)$ arbitrarily as $e_1, e_2, \dots, e_h$ where $h = |D(v)|$. We define a weight function $w : E \to \{1, 2, \dots, h+1\}$ where $w(e_i) = i$ and $w(e) = h + 1$ for all $e \in E \setminus D(v)$.

Given a set of weights ρ on the edges, for each pair (s, t) of terminals we define $\beta_\rho(s, t)$ as the maximum weight a such that the element-connectivity between s and t remains the same even if we remove all edges with weight less than a. We call $\beta_\rho(s, t)$ as the *bottleneck weight* for (s, t). Suppose we used the weight function as defined above based on the numbering for $D(v)$ then v can be eliminated iff $\beta_w(s, t) > h$ for all pairs of terminals (s, t). In fact we can also obtain information on which of the edges in $D(v)$ can be contracted if v cannot be eliminated. Even further, we need to check only the terminal pairs that correspond to edges of a Gomory-Hu tree for the element-connectivity of T. This is captured in the following theorem which forms the crux of our algorithm.

Theorem 7. *Let (R, c) be a Gomory-Hu tree for the element-connectivity of terminal set T in G. Consider an active non-terminal v and the weight function w and let $\ell = \max_{st \in E(R)} \beta_w(s,t)$. Define G' as G/e_ℓ if $\ell < |D(v)| + 1$ and $G - D(v)$ otherwise. Then $\kappa'_{G'}(u,v) = \kappa'_G(u,v)$ for all terminal pairs (u,v).*

Proof. Recall that $D(v) = \{e_1, e_2, \dots, e_h\}$ where $h = |D(v)|$. Let $S = \{e_1, \dots, e_{\ell-1}\}$. Since $\beta_w(s,t) \geq \ell$ for all $st \in E(R)$ it follows that the element-connectivity does not change for any pair $st \in E(R)$ if we delete the edges in S from G. From Lemma 5 it follows in fact that the element-connectivity of all pairs remains the same in $G - S$ as in G. Thus all edges in S are deletable. If $\ell = h + 1$ then $S = D(v)$ and $G' = G - S$ and we have the desired property. If $\ell \leq h$ then there is at least one $st \in E(R)$ such that $\kappa'_{G-S}(s,t) = \kappa'_G(s,t)$ but $\kappa'_{G-S-e_\ell}(s,t) < \kappa'_G(s,t)$. From the reduction lemma applied to $G - S$ we see that e_ℓ is not deletable, and hence by the reduction lemma $(G - S)/e_\ell$ preserves all the element-connectivities of the terminals. This also implies that G/e_ℓ preserves all element-connectivities.

Computing $\beta_w(s,t)$ is relatively fast if we already have an existing maximum flow from s to t. This is captured by the lemma below.

Lemma 3. *Given a maximum s-t-flow f_{st} in $\tilde{G}$, an active non-terminal v and a weighting w, we can find $\beta_w(s,t)$ and a corresponding flow in $O(m)$ time.*

Proof. Consider a maximum s-t flow f_{st} in $\tilde{G}$. In a flow decomposition of f_{st} there is at most one flow path that uses the non-terminal v. We can find such a flow path in $O(m)$ time and reduce the flow by at most one unit to obtain a new flow f'_{st} which does not have any flow through v. Not that f'_{st} is non-zero only on edges e with $w(e) = |D(v)| + 1$. If the value of f'_{st} is the same as that of f_{st} then $\beta_w(s,t) = |D(v)| + 1$ and we are done. Otherwise, we claim that $\beta_w(s,t) = \ell$ iff the maximum bottleneck weight for a path from s to t in the residual graph of f'_{st} in $\tilde{G}$ is ℓ. Assuming that the claim is true we can find $\beta_w(s,t)$ by a maximum bottleneck path computation in the residual graph in $O(m)$ time since the edges are sorted by weight (the algorithm is quite simple but we refer the reader to [13]).

Now we prove the claim. If there is a path of maximum bottleneck weight ℓ in the residual graph we can augment f'_{st} by one unit to obtain a maximum flow f''_{st} that uses only edges with weight ℓ or greater and hence $\beta_w(s,t) \geq \ell$. Suppose $\beta_w(s,t) = \ell'$. Remove the edges $\{e_1, \dots, e_{\ell'-1}\}$ from G and their corresponding arcs from $\tilde{G}$. There is a maximum s-t flow of value $\kappa'_G(s,t)$ in this new graph H. f'_{st} is a flow of value $\kappa'_G(s,t) - 1$ in H and hence there must be an augmenting path in the residual graph of f'_{st} in H and this augmenting path has bottleneck weight at least ℓ' and is also a valid path in residual graph of f'_{st} in $\tilde{G}$. Thus $\ell \geq \ell'$. Thus $\beta_w(s,t) = \ell$ as desired.

Theorem 7 and Lemma 3 lead to an algorithm as follows. We initially compute and then maintain a Gomory-Hu tree (R, c) for C_G on the terminals. For each edge $st \in E(R)$ we also maintain a maximum flow f_{st} in the current graph $\tilde{G}$. In each iteration we do an elimination procedure on an active vertex using Theorem 7. Each iteration either reduces the number of active vertices by one or contracts an edge. Thus the number of iterations is $O(n)$. Algorithm 1 gives a formal description. Note that

Algorithm 1: Reduce a graph G with terminal vertices T

```
   Input: undirected graph $G$, terminals $T$
   // Preprocessing
1  $(R, c) \leftarrow$ Gomory-Hu tree of $C_G$
2  foreach $st \in E(R)$ do
3      $f_{st} \leftarrow$ maximum $st$-flow in $\tilde{G}$
   // Sequence of eliminations
4  while there exists an active non-terminal vertex $v$ do
5      $w \leftarrow$ assign weights to $D(v)$
6      foreach $st \in E(R)$ do
7          compute $\beta_w(s, t)$ in $O(m)$ time using $f_{st}$
8      $\ell \leftarrow \min\{\beta_w(s, t) \mid st \in E(R)\}$
9      if $\ell > |D(v)|$ then
10         $G \leftarrow G - D(v)$
11     else
12         $G \leftarrow G/e_\ell$
13     foreach $st \in E(R)$ do
14         update $f_{st}$ in $O(m)$ time
15 return $G$
```

the tree (R, c) does not change throughout the algorithm but the flows f_{st} for each $st \in E(R)$ get updated in each iteration. We need to clarify how we update these flows, analyze the overall running time, and argue about the correctness of the algorithm.

Updating the flows: Each elimination changes the graph. The algorithm updates the maximum flows f_{st} for each $st \in E(R)$ to reflect this change. If the new graph is $G - D(v)$, then no flow need updating since the computation of $\beta_w(s, t)$ already finds a new flow that avoids all edges in $D(v)$. We address the case when we contract an edge from $D(v)$.

Lemma 4. *Let G be a graph with m edges with terminals T. Let $H = G/e$ for some reducible edge e. If f_{st} is a maximum s-t flow in $\tilde{G}$, then we can find f'_{st}, a maximum s-t-flow in $\tilde{H}$ in $O(m)$ time.*

Proof. Let $e = pq$. We delete flow paths in f_{st} that use p or q. This removes at most 2 unit of flow (since each non-terminal has unit capacity) and the reduced flow is a valid flow in $\tilde{H}$. In two augmentations we can find a maximum flow in $\tilde{H}$. Each step can be easily implemented in $O(m)$ time.

Analysis of running time: The time spent to build the Gomory-Hu tree (R, c) and the initial maximum flows for each edge $st \in E(R)$ take $|T| - 1$ maximum flow computations. Thus the time for this is $O(|T| \text{ MF}(n, m))$

As we argued there are $O(n)$ iterations of the while loop. In each iteration we need to compute $\beta_w(st)$ for each $st \in E(R)$. Lemma 3 shows that each such computation can be done in $O(m)$ time using the stored maximum flow; thus the total time is $O(|T|m)$. Updating the maximum flows also takes $O(|T|m)$ time using Lemma 4.

Thus the overall running time for all the iterations of the while loop is $O(|T|nm)$ which dominates the time to compute the initial Gomory-Hu tree.

Correctness: Theorem 7 shows the correctness of the elimination procedure. It remains to argue that the Gomory-Hu tree (R, c) computed in the preprocessing step remains valid throughout the algorithm as the graph G changes and gets simplified. A similar idea is used implicitly by Gabow for preserving local edge-connectivity while applying split-off operations [11]. The following simple lemma gives a justification.

Lemma 5. *Let (R, c) be a Gomory-Hu tree for C_G with terminal vertices T in G. Let H be a reduction of G. If $\kappa'_G(s,t) = \kappa'_H(s,t)$ for all $st \in E(R)$ then (R, c) is also a Gomory-Hu tree for C_H.*

5 Proof of reduction lemma via setpairs

In this section we give a new proof of Theorem 2 via the use of setpairs. We will assume without loss of generality that T is an independent set by sub-dividing each edge between terminals. First we introduce some functions over setpairs over the vertex set V. These functions and their properties have been explored in papers on network design for element connectivity [19,9,5] and in particular we refer the reader to [5] for relevant proofs.

Let $\mathcal{S}$ be set of all setpairs over vertex set V of the given graph G. Recall that T is the set of terminals. We define g, ℓ, f on $\mathcal{S}$ as follows. For a set pair $W = (W_t, W_h)$,

$$g(W) = \max\{\kappa'_G(u,v) : u \in W_t \cap T, v \in W_h \cap T\}$$

Here the $\max$ follows the convention $\max \emptyset = 0$.

$$\ell(W) = |V - W_t - W_h|$$

$$f(W) = g(W) - \ell(W)$$

The lemmas below are known from prior work; see [9,5].

Lemma 6. *The following properties hold: (i) g is symmetric skew-bisupermodular, (ii) ℓ is symmetric bimodular, (iii) f is symmetric skew-bisupermodular, and (iv) $|\delta(W)|$ is symmetric bi-submodular.*

A setpair W is *tight* if $f(W) = |\delta(W)|$. The lemma below follows from skew-bisupermodularity of f and symmetric bi-submodularity of $|\delta()|$.

Lemma 7 (Lemma 3.1 [5]). *If W and Y are tight, then one of the following holds:*

1. *$W \oplus Y$ and $W \otimes Y$ are tight.*
2. *$\overline{W} \oplus Y$ and $\overline{W} \otimes Y$ are tight.*

We are now ready to prove the Reduction Lemma.

Proof. Consider any edge pq between non-terminals and graphs $G_1 = G - pq$ and $G_2 = G \setminus pq$. Suppose the lemma is false. There must be distinct [3] terminal pairs (s,t) and (x,y) such that $\kappa'_{G_1}(s,t) < \kappa'_G(s,t)$ *and* $\kappa'_{G_2}(x,y) < \kappa'_G(x,y)$.

[3] It is not hard to see that for any (s,t), $\kappa'_G(s,t) = \kappa'_{G_1}(s,t)$ or $\kappa'_G(s,t) = \kappa'_{G_2}(s,t)$.

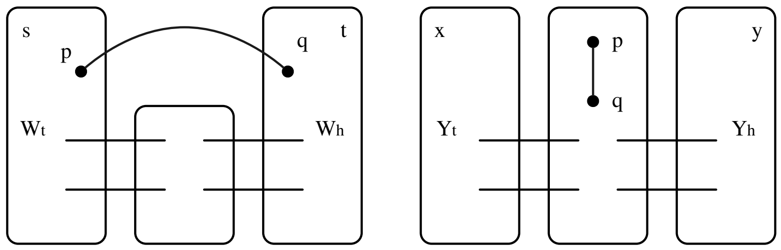

Fig. 3. (left) $\kappa'_{G_1}(s,t) = \kappa'_G(s,t)$ when pq is removed. (right) $\kappa'_{G_2}(x,y) = \kappa'_G(x,y) - 1$ when pq is contracted. Recall that T is an independent set.

For the pair (s,t), since the connectivity decreased when pq was deleted, there exists a setpair W such that $s, p \in W_t, t, q \in W_h$, $\ell(W) = \kappa'_G(s,t)-1$ and $\delta(W) = \{pq\}$; see Fig. 3. If we delete pq, then $V - W_t - W_h$ becomes a valid cut-set that separates s and t with smaller value. The cut sets contain only non-terminals because T is an independent set. For the pair (x,y), since connectivity decreased when pq was contracted, there exists a setpair Y such that $x \in Y_t, y \in Y_h, p, q \in V - Y_t - Y_h$, $\ell(Y) = \kappa'_G(x,y)$ and $\delta(Y) = \emptyset$; see Fig. 3. If we contract pq to form p', $V - Y_t - Y_h - p'$ becomes a smaller cut-set.

We have $f(W) = g(W) - \ell(W) = \kappa'_G(s,t) - (\kappa'_G(s,t) - 1) = 1$, and $f(Y) = g(Y) - \ell(Y) = \kappa'_G(x,y) - \kappa'_G(x,y) = 0$. Because $|\delta(W)| = 1$ and $|\delta(Y)| = 0$, W and Y are tight. We now use Lemma 7 to consider two cases.

Suppose $W \oplus Y$ and $W \otimes Y$ are tight. We have $f(W \oplus Y) = |\delta(W \oplus Y)|$ and $f(W \otimes Y) = |\delta(W \otimes Y)|$. If $\delta(W \oplus Y)$ is not empty, then either $\delta(W) - \{pq\}$ or $\delta(Y)$ must not be empty, which cannot be since $|\delta(W) - pq| = 0$ and $|\delta(Y)| = 0$. (We exclude pq from $\delta(W)$ because $pq \notin \delta(W \oplus Y)$.) Therefore, $\delta(W \oplus Y)$ is empty. Similarly, $\delta(W \otimes Y)$ must be empty. Therefore, $f(W) + f(Y) = 1 + 0 > 0 + 0 = f(W \oplus Y) + f(W \otimes Y)$, which is a contradiction since f is bisupermodular. A similar argument can be applied to the case when $\overline{W} \oplus Y$ and $\overline{W} \otimes Y$ are tight by exchanging W with $\overline{W}$.

Acknowledgements. This work was supported in part by Chandra Chekuri's NSF grants CCF-1319376 and CCF-1526799, and Jeff Erickson's NSF grant CCF-0915519.

References

1. Aazami, A., Cheriyan, J., Jampani, K.: Approximation Algorithms and Hardness Results for Packing Element-Disjoint Steiner Trees in Planar Graphs. Algorithmica 63(1–2), 425–456 (2012)
2. Benczúr, A.A.: Counterexamples for directed and node capacitated cut-trees. SIAM Journal on Computing 24(3), 505–510 (1995)
3. Chekuri, C., Korula, N.: A graph reduction step preserving element-connectivity and packing steiner trees and forests. SIAM Journal on Discrete Mathematics 28(2), 577–597 (2014)
4. Cheriyan, J., Salavatipour, M.: Packing Element-Disjoint Steiner Trees. ACM Transactions on Algorithms 3(4) (2007).
5. Cheriyan, J., Vempala, S., Vetta, A.: Network Design via Iterative Rounding of Setpair Relaxations. Combinatorica 26(3), 255–275 (2006)

6. Cheung, H.Y., Lau, L.C., Leung, K.M.: Graph connectivities, network coding, and expander graphs. In: Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science. pp. 190–199. FOCS '11 (2011)
7. Duan, R.: Breaking the $O(n^{2.5})$ Deterministic Time Barrier for Undirected Unit-Capacity Maximum Flow. In: Proceedings of ACM-SIAM SODA, pp. 1171–1179. (2013)
8. Even, S., Tarjan, R.: Network flow and testing graph connectivity. SIAM Journal on Computing 4(4), 507–518 (1975)
9. Fleischer, L., Jain, K., Williamson, D.: Iterative Rounding 2-Approximation Algorithms for Minimum-Cost Vertex Connectivity Problems. Journal of Computer and System Sciences 72(5), 838–867 (2006)
10. Frank, A., Ibaraki, T., Nagamochi, H.: On sparse subgraphs preserving connectivity properties. Journal of Graph Theory 17(3), 275–281 (1993)
11. Gabow, H.N.: Efficient splitting off algorithms for graphs. In: Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing. pp. 696–705. STOC '94, ACM, New York, NY, USA (1994)
12. Gabow, H.N.: Using expander graphs to find vertex connectivity. In: Proc. 41st Annual IEEE Symposium on Foundations of Computer Science. pp. 410–420 (2000)
13. Gabow, H.N., Tarjan, R.E.: Algorithms for Two Bottleneck Optimization Problems. Journal of Algorithms 9(3), 411–417 (1988)
14. Gabow, H., Sankowski, P.: Algebraic algorithms for b-matching, shortest undirected paths, and f-factors. In: Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on. pp. 137–146 (Oct 2013)
15. Hao, J., Orlin, J.B.: A faster algorithm for finding the minimum cut in a graph. In: Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 165–174. SODA '92, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (1992)
16. Hariharan, R., Kavitha, T., Panigrahi, D., Bhalgat, A.: An $\tilde{O}(mn)$ Gomory-Hu tree construction algorithm for unweighted graphs. In: Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing. pp. 605–614. STOC '07, ACM (2007)
17. Henzinger, M.R., Rao, S., Gabow, H.N.: Computing vertex connectivity: New bounds from old techniques. Journal of Algorithms 34(2), 222 – 250 (2000)
18. Hind, H., Oellermann, O.: Menger-Type Results for Three or More Vertices. Congressus Numerantium pp. 179–204 (1996)
19. Jain, K., Măndoiu, I., Vazirani, V., Williamson, D.: A Primal-Dual Schema Based Approximation Algorithm for the Element Connectivity Problem. In: Proceedings of the 10th annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 484–489 (1999)
20. Karger, D.: Random Sampling in Graph Optimization Problems. Ph.D. thesis, Stanford University (1994)
21. Kawarabayashi, K., Thorup, M.: Deterministic Global Minimum Cut of a Simple Graph in Near-Linear Time. In: Proceedings of ACM STOC, pp. 665–674. (2015)
22. Klimmek, R., Wagner, F.: A simple hypergraph min cut algorithm. Tech. Rep. B 96-02, Bericht FU Berlin Fachbereich Mathematik und Informatik (1996)
23. Lau, L.C., Yung, C.K.: Efficient edge splitting-off algorithms maintaining all-pairs edge-connectivities. SIAM Journal on Computing 42(3), 1185–1200 (2013)
24. Lovász, L.: On Some Connectivity Properties of Eulerian Graphs. Acta Mathematica Hungarica 28(1), 129–138 (1976)
25. Mader, W.: A Reduction Method for Edge-Connectivity in Graphs. Annals of Discrete Mathematics 3, 145–164 (1978)
26. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency. Springer Verlag, Berlin Heidelberg (2003)