Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Discrete Applied Mathematics 127 (2003) 679–689
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

Parametric min-cuts analysis in a network☆

Y.P. Aneja[a], R. Chandrasekaran[b], K.P.K. Nair[c]

[a] *Faculty of Business Administration, University of Windsor, Windsor, Ont., Canada N9B 3P4*
[b] *School of Computer Science, University of Texas at Dallas, Richardson, TX, USA*
[c] *Faculty of Administration, University of New Brunswick, Fredericton, N.B., Canada E3B5A3*

Received 10 July 2001; received in revised form 24 April 2002; accepted 10 June 2002

Abstract

The all pairs minimum cuts problem in a capacitated undirected network is well known. Gomory and Hu showed that the all pairs minimum cuts are revealed by a min-cut tree that can be obtained by solving exactly $(n-1)$ maximum flow problems, where n is the number of nodes in the network.

In this paper we consider first the problem of finding parametric min-cuts for a specified pair of nodes when the capacity of an arc i is given by $\min\{b_i, \lambda\}$, where λ is the parameter, ranging from 0 to ∞. Next we seek the parametric min-cuts for all pairs of nodes, and achieve this by constructing min-cut trees for at most $2m$ different values of λ, where m is the number of edges in the network.

Keywords: Multiterminal maximal flows; Parametric analysis; Combinatorial optimization

1. Introduction

Consider an undirected network $G(N, A, b)$ with node set N ($|N| = n$), arc set $A(|A| = m)$ and a capacity vector b such that arc $i \in A$ has capacity b_i. The "all pairs minimum cuts" problem in such a network has been solved efficiently by Gomory and Hu [8]. They showed that there are at most $(n-1)$ distinct min-cut values, and by solving exactly $(n-1)$ maximum flow problems one can construct a so-called *min-cut tree* T,

☆ This work was supported by research grants to Y.P. Aneja and K.P.K. Nair from the Natural Sciences and Engineering Research council of Canada.
E-mail address: aneja@uwindsor.ca (Y.P. Aneja).

0166-218X/03/$ - see front matter
PII: S0166-218X(02)00496-1

where value on a link between nodes x and y in T, is the value of the minimum cut separating x and y. Furthermore, one such min-cut is given by the partition of node set N obtained by removing this link from T. Min-cut between any two nodes x and y of N is given by the smallest value link on the unique path between x and y in T. An elegant exposition of the above is available in Ford and Fulkerson [6]. It should be noted that a *link* between nodes x and y on a min-cut tree has a different meaning from that of an arc between nodes x and y in the network. For this reason the terms *link* and *arc* will be used appropriately.

Parametric analyses of optimization problems are of interest in many ways [1]. Relating to networks, parametric analyses of spanning tree, shortest path, and maximum flow problems have received considerable attention [5,9]. Relating to maximum flows, the parametrization if restricted to linear capacity and only for arcs at source and sink, the problem has been shown to admit efficient solution [7].

The parametric analysis that we consider can be thought of as a bottle-neck type in the sense that the parametric capacity of arc $i \in A$ is $\min\{b_i, \lambda\}$, with parameter λ ranging from 0 to ∞. In Section 2 we consider the parametric min-cuts problem for a specified pair of nodes and present a strongly polynomial algorithm to compute its solution. The algorithm is a generalization of Newton's method in the context of combinatorial optimization. In several combinatorial problems analogous specializations have been used [3,4,11]. An example is provided for illustrating the algorithm.

In Section 3 we consider the problem of finding parametric min-cuts between all pairs of nodes. It is shown that such min-cuts can be obtained by solving *min-cut tree* problems for at most $2m$ different values of λ. Using this result a strongly polynomial algorithm is presented and illustrated with an example. Finally in Section 4 we outline some applications of this work.

2. Parametric min-cuts for a specified $s-t$ pair

Given parametric capacity $b_i(\lambda) = \min\{b_i, \lambda\}$ for each arc $i = 1, \ldots, m$, let the network G be denoted by $G(\lambda)$ and the specified source sink pair by (s, t). A cut H separating s and t is a set of arcs $(X, \bar{X})$ with $s \in X$ and $t \in \bar{X}$. To simplify notation, we will simply represent this cut by the node set X or $\bar{X}$. The problem is to identify a spectrum of minimum cuts separating s and t as λ increases from zero. Let $v(\lambda)$ be the value of a minimum cut, separating s and t in $G(\lambda)$, expressed as a function of λ. Obviously $v(\lambda)$ is also the maximum flow value from s to t.

Lemma 1. *The function $v(\lambda)$ is piecewise linear concave with integral slope for each linear piece and has at most $(n-1)$ breakpoints.*

Proof. Take any $s-t$ cut H. Let $h(\lambda) = \sum_{i \in H} b_i(\lambda)$. Since each $b_i(\lambda)$ is piecewise linear concave in λ, so is $h(\lambda)$ and represents the capacity of the cut H in $G(\lambda)$, Slope of any linear piece in $h(\lambda)$, for a given λ, is an integer with value equal to the number of arcs with capacity λ. Since $v(\lambda) = \min\{h(\lambda) : H \in \Omega\}$, where Ω is the set of all

$s-t$ cuts, $v(\lambda)$ is piecewise linear concave with integral slopes. Since the largest slope (slope of the first linear piece of $v(\lambda)$) gives the cardinality of an $s-t$ cut with minimal cardinality, it is bounded by $(n-1)$. Further, there must be a drop of at least one unit of slope between two consecutive linear pieces of $v(\lambda)$. Hence $(n-1)$ provides an upper bound for the number of breakpoints in $v(\lambda)$. □

It is important to note that there are two types of breakpoints in $v(\lambda)$. Each linear piece of $v(\lambda)$ corresponds to the capacity of some $s-t$ min-cut $D(\lambda)$ in $G(\lambda)$. However, two adjacent linear pieces of $v(\lambda)$ may correspond to the same cut and the breakpoint between these two pieces then occurs due to saturation of some arc(s) at that breakpoint. By saturation of an arc i we mean that λ becomes equal to b_i, the capacity of that arc.

Our objective is to generate $v(\lambda)$ by identifying all its breakpoints. To describe the algorithm, we need to introduce certain sets and definitions. For a given λ, let $D(\lambda)$ be a min-cut (separating s and t) with value $v(\lambda)$ in $G(\lambda)$, and $q(\lambda)$ denote the number of arcs at capacity λ in $D(\lambda)$. Let $\lambda_{\max}=\max\{b_i\colon i\in A\}$. Choose λ_ε such that $0<\lambda_\varepsilon<\min\{b_i\colon i\in A\}$. Then clearly $v(\lambda_{\max})=v^*$, the maximum flow value in G, and $D(\lambda_\varepsilon)$ is a cut in G with minimum cardinality.

Let E denote the vector of equations in the λv-plane, sorted in descending order of their slopes. The rth element of E defines the equation $E_r : v_r(\lambda)=v(\lambda_r)+q(\lambda_r).(\lambda-\lambda_r)$, and is stored as $((\lambda_r,v(\lambda_r)),q(\lambda_r),D(\lambda_r))$. Let R denote the vector of breakpoints of $v(\lambda)$ stored in ascending order of λ-values and stored as $[\{(\lambda_1,v(\lambda_1));D_1\},\{(\lambda_2,v(\lambda_2));D_2\},\ldots,\{(\lambda_p,v(\lambda_p));D_p\}]$. Here D_r is a min-cut for $G(\lambda)$ in the λ-interval $[\lambda_{r-1}\lambda_r]$, assuming $\lambda_0\equiv 0$. For $\lambda\geqslant\lambda_p$, any min-cut in G is a min-cut in $G(\lambda)$. Each breakpoint of $v(\lambda)$ is determined by the intersection of two consecutive line segments of $v(\lambda)$. Equations in set F define the initial segment of $v(\lambda)$, and define all the breakpoints generated so far.

At the start of the algorithm both E,F and R are null vectors. At termination, R contains all the distinct breakpoints of $v(\lambda)$, and uniquely defines the function $v(\lambda)$. Similarly, at termination F provides the set of equations that also define $v(\lambda)$ uniquely.

Algorithm. *Parametric Min-Cuts*

Step 0: Find $v(\lambda_\varepsilon),q(\lambda_\varepsilon),D(\lambda_\varepsilon),v(\lambda_{\max})$ and $D(\lambda_{\max})$. Let $E=[((0,0),q(\lambda_\varepsilon),D(\lambda_\varepsilon)),((\lambda_{\max},v^*),0,D(\lambda_{\max}))]$, Let $\hat{D}\leftarrow D(\lambda_\varepsilon)$, $R\leftarrow\emptyset$, and $F\leftarrow\emptyset$.

Step 1: Let (λ_0,v_0) be the intersection point of the first two equations in E. If for $\lambda=\lambda_0$, a min-cut has been found earlier then insert $((\lambda_0,v_0),\hat{D})$ as the last element of R and go to step 4, else go to step 2.

Step 2: If the slope of the first equation is exactly one more than that of the second, then insert $((\lambda_0,v_0),\hat{D})$ as the last element in R and go to step 4, else go to step 3.

Step 3: Find $v(\lambda_0)$ and $D(\lambda_0)$. If $v(\lambda_0)=v_0$ then insert $((\lambda_0,v_0);\hat{D})$ as the last element of R, let $\hat{D}\leftarrow D(\lambda_0)$, and go to step 4, else insert equation represented by $((\lambda_0,v(\lambda_0)),q(\lambda_0),D(\lambda_0))$ between the first two equations in E and go to step 1.

Step 4: Remove the first equation from E and store it as the last element in F. If $E=[((\lambda_{\max},v^*),0,D(\lambda_{\max}))]$, then store it as the last element in F and stop, else go to step 1.

Let us make some observations about the algorithm. At step 0, the first line in E corresponds to a minimum cardinality cut, its slope is no more than $(n-1)$, the maximum degree of node s. Successive lines generated at step 3 have strictly decreasing integral slopes. Each time a breakpoint is discovered, cardinality of E decreases by one. Step 1 handles a degenerate situation where at a given breakpoint λ_0, there are more than two cuts optimal in $G(\lambda_0)$. In step 2, when the slope of the first two lines in R differs by one, their intersection point must be a breakpoint. Hence the algorithm stops in no more than $q(\lambda_\varepsilon)$ iterations. Step 3 identifies breakpoints whose two adjacent linear pieces have slopes that differ by more than one unit. In fact it is easy to show that the exact number of max-flow problems that need to be solved equals $(1+\min\{q(\lambda_\varepsilon), 2p\})$, where p is the number of breakpoints of $v(\lambda)$.

The algorithm enters breakpoints in R in ascending order of λ-values, and terminates with all the breakpoints of $v(\lambda)$. Intuitively, in step 1, if $v_0 = v(\lambda_0)$ then the first equation identifies a linear piece of $v(\lambda)$.

The validity of the algorithm follows from the discussions in [3,4,11]. A brief proof is given below.

Lemma 2. *At termination of the above algorithm, set F correctly defines $v(\lambda)$.*

Proof. Recall from Lemma 1 that $v(\lambda) = \min\{h(\lambda) : H \in \Omega\}$, where $h(\lambda)$ is the cut-capacity of H in $G(\lambda)$. Hence each $h(\lambda)$ acts as an upper bound for $v(\lambda)$. Also, since $h(\lambda)$ is piecewise linear concave, its every linear piece also provides an upper bound for $v(\lambda)$. Since each equation in E, and hence in F, corresponds to some cut, we have $v(\lambda) \leqslant \min\{v_r(\lambda)\colon r = 1, \ldots, |F|\}$. Further, if for some $\lambda_1 < \lambda_2$, both points $(\lambda_1, v(\lambda_1))$ and $(\lambda_2, v(\lambda_2))$, correspond to the same linear equation in E, then the entire line segment joining these two points defines a part of $v(\lambda)$. In this case (step 3), the intersection point of the first two equations in E is recorded as a breakpoint of $v(\lambda)$, and the first equation in E is moved to F. Further if two consecutive equations in E differ in slopes by exactly one then the intersection point of these two lines must be a breakpoint of $v(\lambda)$, and the first equation in E is moved to F. At termination, therefore, equations in F define the complete $v(\lambda)$ function and $v(\lambda) = \min\{v_r(\lambda)\colon r = 1, \ldots, |F|\}$. $\square$

The following example illustrates the algorithm. Consider the network in Fig. 1 with the arc capacities as shown. Let the source sink pair be (2,5).

Here $\lambda_{\max} = 7$, and let $\lambda_\varepsilon = 0.5$. Then $v(\lambda_{\max}) = v^* = 11$, the maximum flow value with a min-cut given by $D(\lambda_{\max}) = \{2\}$. As mentioned earlier, the cut $\{2\}$ represents the set of arcs in the set $(X, \bar{X})$, where $X = \{2\}$. Also, $v(\lambda_\varepsilon) = 4(0.5) = 2$ with a min-cut (a minimum cardinality cut) given by $D(\lambda_\varepsilon) = \{1,2,3,4,6\} = \{5\}$, and $q(\lambda_\varepsilon) = 4$.

The algorithm *Parametric Min-Cuts* proceeds as follows:

Iteration 1:

Step 0: $E = [((0,0),4,\{5\}),((7,11),0,\{2\})]$, $\hat{D} = \{5\}$, $R \leftarrow \emptyset, F \leftarrow \emptyset$.

Step 1: $(\lambda_0, v_0) = (11/4, 11)$—the intersection point of the two equations in E.

Step 3: Now $v(\lambda_0) = 9.5$, $D_0 = \{2\}$ provides a min-cut, and $q(\lambda_0) = 2$. Since $v(\lambda_0) \neq v_0$, we insert $((11/4, 9.5), 2, \{2\})$ between the two equations of E. So $E = [((0,0),4,\{5\}),((11/4,9.5),2,\{2\}),((7,11),0,\{2\})]$.

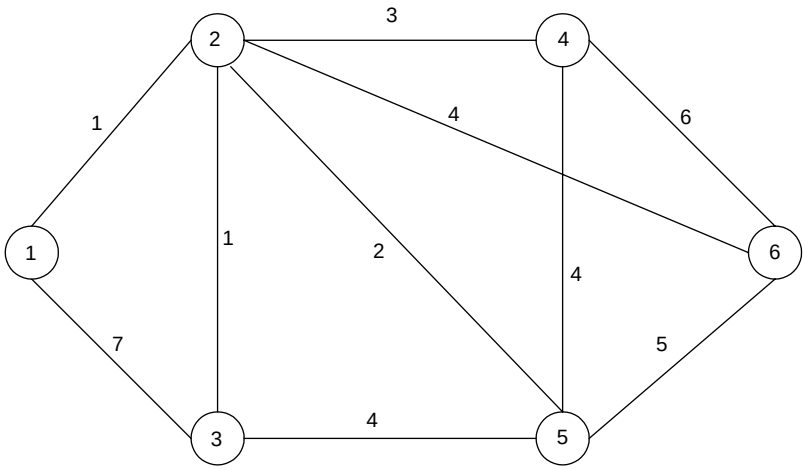

Fig. 1. An example network.

Iteration 2:

Step 1: $(\lambda_0, v_0) = (2, 8)$—the intersection point of the first two equations: $v = 4\lambda$ and $v = 4 + 2\lambda$.

Step 3: $v(\lambda_0) = 8 = v_0$, with $D_0 = \{2\}$. Hence $(2, 8)$ is a breakpoint of $v(\lambda)$. We update E and R so that $E = [((11/4, 9.5), 2, \{2\}), ((7, 11), 0, \{2\})]$, $R = [((2, 8); \{5\}]$, and $F = [((0, 0), 4, \{5\})]$.

Iteration 3:

Step 1: $(\lambda_0, v_0) = (3.5, 11)$.

Step 3: $v(\lambda_0) = 10.5$, with $D_0 = \{2\}$ with $q(\lambda_0) = 1$. Since $v(\lambda_0) < 11 = v_0$, E is updated to

$$E = [((11/4, 9.5), 2, \{2\}), ((3.5, 10.5), 1, \{2\}), ((7, 11), 0, \{2\})].$$

Iteration 4:

Step 1: $(\lambda_0, v_0) = (3, 10)$.

Step 2: Since slope difference between the first two equations is exactly 1, $(3, 10)$ is the next breakpoint of $v(\lambda)$. After updating

$R = [((2, 8); \{5\}), ((3, 10); \{2\})]$, $E = [((3.5, 10.5), 1, \{2\}), ((7, 11), 0, \{2\})]$, and $F = [((0, 0), 4, \{5\}), ((11/4, 9.5), 2, \{2\})]$.

Iteration 5:

Step 1: $(\lambda_0, v_0) = (4, 11)$.

Step 2: Since slope difference between the first two equations is exactly 1, $(4, 11)$ is the next breakpoint of $v(\lambda)$. After updating

$R = [((2, 8); \{5\}, ((3, 10); \{2\}), ((4, 11); \{2\})]$ $E = [((7, 11), 0, \{2\})]$ and $F = [((0, 0), 4, \{5\}), ((11/4, 9.5), 2, \{2\}), ((3.5, 10.5), 1, \{2\})]$.

Step 4: Termination criterion is met and the algorithm stops with $F = [((0, 0), 4, \{5\}), ((11/4, 9.5), 2, \{2\}), ((3.5, 10.5), 1, \{2\}), ((7, 11), 0, \{2\})]$.

The function $v(\lambda)$ has three breakpoints listed in R. For $\lambda \in [0-2]$, $D=\{5\}$ provides a min-cut in $G(\lambda)$, and for $\lambda \in [2-4]$, a min-cut in $G(\lambda)$ is given by $D=\{2\}$. Since $D(\lambda_{\max})=\{2\}$ is a min-cut in G, $\{2\}$ is also a min-cut for $\lambda > 4$.

Thus, although $v(\lambda)$ has three breakpoints with four linear pieces, there are only two min-cuts needed to define the function. Equivalently, there is only one change in the parametric min-cut and this occurs at $\lambda = 2$.

It should be noted that since the algorithm generates breakpoints in increasing order of λ, max flow in $G(\lambda_1)$ is a feasible flow for $G(\lambda_2)$ for $\lambda_1 < \lambda_2$.Thus the successive max flow problems can use previous max flows for reducing computational effort.

3. Parametric min-cut trees

Recall the min cut-tree problem in an undirected network [6,8], mentioned earlier. Here we consider a parametric version of the problem in which the parametric capacity of arc $i \in A$ is $\min\{b_i, \lambda\}$. We wish to find a spectrum of min-cut trees in $G(\lambda)$ as λ varies from 0 to infinity. Each such parametric min-cut tree will be a min-cut tree in $G(\lambda)$ for an interval of λ. Before we describe an algorithm to find a spectrum, we want to establish a bound on the number of different min-cut trees that will be generated as λ varies over $\Re^+$. Since, for a given source–destination pair, $v(\lambda)$ can have at most $(n-1)$ breakpoints, entire $\Re^+$ line is divided into at most n intervals by this pair of nodes. As there are $\binom{n}{2}$ pairs of nodes, the breakpoints for $\binom{n}{2}$ different $v(\lambda)$ functions super imposed divide $\Re^+$ into at most $(n^2(n-1)/2)$ intervals. Clearly, the min-cut tree of $G(\lambda)$ does not change as λ ranges over any such interval. Hence $(n^2(n-1)/2)$ is an upper bound on the number of different min-cut trees that the spectrum would contain. We show below that this bound is very loose, and that the maximum number of different min-cut trees is less than $2m$, m being the number of arcs in the network.

Let $T_1, T_2, \ldots, T_L$ denote the different min cut-trees of $G(\lambda)$ as λ increases. Assume the first cut-tree T_1 is a cut-tree in $G(\lambda_\varepsilon)$, where, as defined earlier, λ_ε is chosen such that $0 < \lambda_\varepsilon < \min\{b_i\colon i \in A\}$. Denoted by $\lambda_1, \lambda_2, \ldots, \lambda_{L-1}$ the min cut-tree change points. Thus $[\lambda_\ell \lambda_{\ell+1}]$ is the interval of λ for which $T_{\ell+1}$ is a min cut-tree in $G(\lambda)$, noting that $\lambda_0 \equiv 0$ and $\lambda_L \equiv \infty$. Also note that both T_ℓ and $T_{\ell+1}$ are min cut-trees in $G(\lambda_\ell)$.

Consider the tree T_ℓ. If it has a link g that connects nodes x and y in T_ℓ, then, as mentioned earlier, removal of this link provides a partition of the node set N and thereby reveals a min-cut separating x and y in $G(\lambda)$ for $\lambda \in [\lambda_{\ell-1}\lambda_\ell]$. As established in the previous section, the function $v(\lambda)$ for this cut is a piecewise linear concave function on $\Re^+$, with integral slope for each linear piece. Let s_g be the initial slope of $v(\lambda)$ in the open interval $(\lambda_{\ell-1}\lambda_\ell)$ and $S_\ell = \sum_{g \in T_\ell} s_g$.

Lemma 3. $S_1 > S_2 > \cdots > S_L$.

Proof. Consider the tree T_ℓ which is a min-cut tree of $G(\lambda)$ for $\lambda \in (\lambda_{\ell-1}\lambda_\ell)$. Each link of this tree represents a cut—a min-cut between any pair of nodes which are disconnected by removal of this link in T_ℓ. Thus, for $\lambda \in (\lambda_{\ell-1}\lambda_\ell)$, $v(\lambda)$, for every pair

of nodes, is defined by a cut represented by one of the links in T_ℓ. Since $T_{\ell+1}$ differs from T_ℓ in at least one of the links, the $v(\lambda)$ function for some pair of nodes must be defined by two different cuts for the two intervals $(\lambda_{\ell-1}\lambda_\ell)$ and $(\lambda_\ell\lambda_{\ell+1})$, resulting in a decrease in initial slopes in $v(\lambda)$ for that pair of nodes, for the two intervals. □

Lemma 4. $S_1 \leqslant 2m - \max_j\{d_j\}$, *where* d_j *is the degree of node* j.

Proof. Recall that T_1 is a min-cut tree in $G(\lambda_\varepsilon)$, and hence provides cuts with minimum cardinality between every pair of nodes. Thus S_1 is the sum of cardinalities of minimum cardinality cuts separating nodes corresponding to the links of T_1. Since $\sum_{j\in N} d_j = 2m$, the proof follows from the fact that the cardinality of a minimum cardinality cut separating a node j from any other node cannot be more than d_j. □

Before we describe the algorithm for generating $T_1, T_2, \ldots, T_L$, consider the $v(\lambda)$ function for a given pair of nodes. As we mentioned in Section 2, $v(\lambda)$ has two kinds of breakpoints—some breakpoints that result in a min-cut change, and others that do not. Consider a breakpoint $\tilde{\lambda}$ that results in a min-cut change. That is, for any $\delta > 0$ and arbitrarily small, any min-cut $D(\tilde{\lambda}_0 - \delta)$ in $G(\tilde{\lambda}_0 - \delta)$ is not a min-cut in $G(\tilde{\lambda}_0 + \delta)$. Note that $D(\tilde{\lambda}_0 - \delta)$ and $D(\tilde{\lambda}_0 + \delta)$ are both min-cuts in $G(\tilde{\lambda}_0)$. In the algorithm below, we will refer to $D(\lambda + \delta)$ as $D(\lambda^+)$, a min-cut in $G(\lambda)$.

Algorithm. *Spectrum*

Step 0: Set $\lambda_0 \leftarrow \lambda_\varepsilon$. Determine T_1, a min-cut tree in $G(\lambda_\varepsilon)$. Set $\ell \leftarrow 1$.

Step 1: For each pair of nodes connected by a link in T_ℓ, record the corresponding $v(\lambda)$ function if it has not been found earlier. Among all the breakpoints of $v(\lambda)$ functions, for the pairs of nodes directly connected by links in T_ℓ, determine $(\lambda_\ell, v(\lambda_\ell))$—the breakpoint with the smallest $\lambda > \lambda_{\ell-1}$ that results in a min-cut change. Terminate if no such λ exits.

Step 2: Starting with the cut $D(\lambda_\ell^+)$, determine a min-cut tree in $G(\lambda_\ell)$. Label this min-cut tree as $T_{\ell+1}$. Set $\ell \leftarrow \ell + 1$, and go to step 1.

Lemma 5. *Algorithm* "*Spectrum*" *determines all min-cut trees in at most* $(2m - \max_j\{d_j\})$ *iterations.*

Proof. Consider a min-cut tree T_ℓ at some iteration of the algorithm. obtained by solving the min-cut tree problem in $G(\lambda_{\ell-1})$. Step 1 determines the smallest λ-value $> \lambda_{\ell-1}$ (say λ_ℓ) such that for some link in T connecting nodes x and y, the min-cut separating x and y in $G(\lambda_\ell)$ is different than the one represented by the current min-cut tree T_ℓ. Thus the S-value of the min-cut tree in $G(\lambda_\ell)$ is less than that of T_ℓ. It must, therefore, be a different min-cut tree than T_ℓ. Since (from Lemma 3) the S-value of the initial cut tree is no more than $2m - \max_j\{d_j\}$, the algorithm would stop in no more that $2m - \max_j\{d_j\}$ iterations. □

As evident from the algorithm, the effort needed will be proportional to the number of parametric min-cut trees. Although the bound established for this number in Lemma

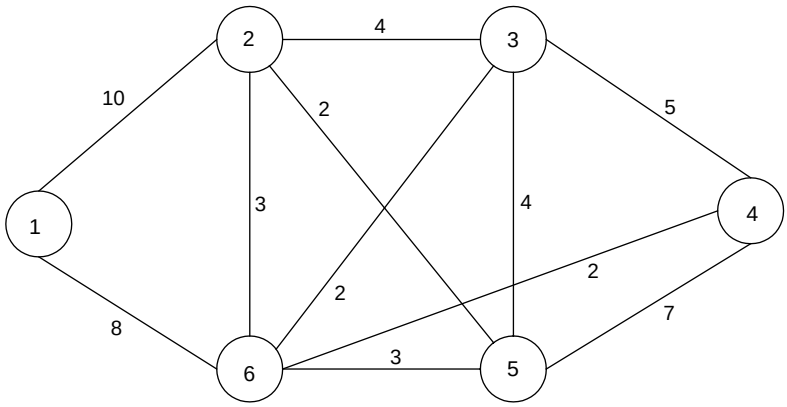

Fig. 2. Network for parametric min-cut trees

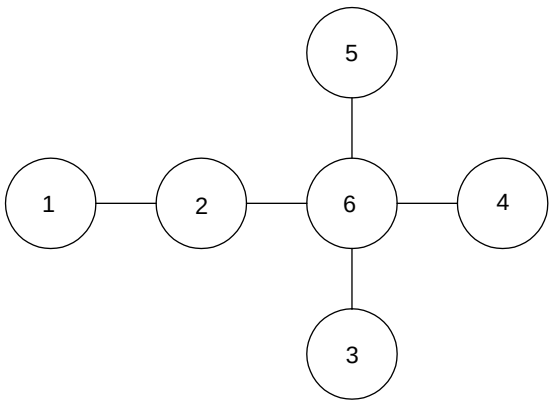

Fig. 3. Min-cut tree T_1

5 is $(2m - \max_j\{d_j\})$, we believe that the actual number will be considerably less. In algorithm "*Spectrum*" also, we may require solving max flow problem in $G(\lambda)$ for several values of λ, for the same source–sink pair. As these λ-values are always increasing, we can take advantage of the previous solutions in such situations.

An Example. Consider the network in Fig. 2.

Iteration number 1:

Step 0: $\lambda_\varepsilon = 1$. Following is a min-cut tree in $G(1)$, labeled as T_1 in Fig. 3.

Step 1: We now use the $v(\lambda)$ function for each link to identify the breakpoints that correspond to a min-cut change:

Link $(1,2)$: Cut $X_1 = \{1\}$ remains optimal for all $\lambda > 0$.

Link $(2,6)$: Cut $X_1 = \{1,2\}$ remains optimal for all $\lambda > 0$.

Link $(4,6)$: Cut $X_1 = \{4\}$ remains optimal for all $\lambda > 0$.

Link $(5,6)$: Cut $X_1 = \{5\}$ optimal for $\lambda \in (0,4]$ and $X_1 = \{3,5,4\}$ optimal for $\lambda \geqslant 4$.

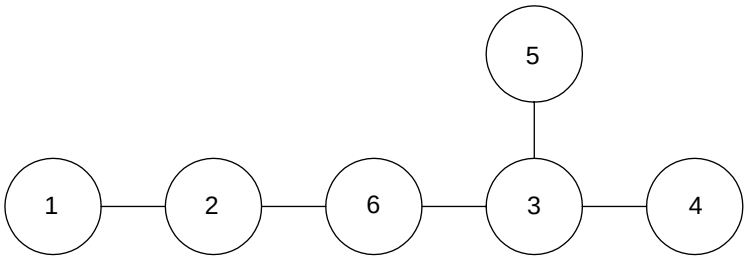

Fig. 4. Min-cut tree T_2.

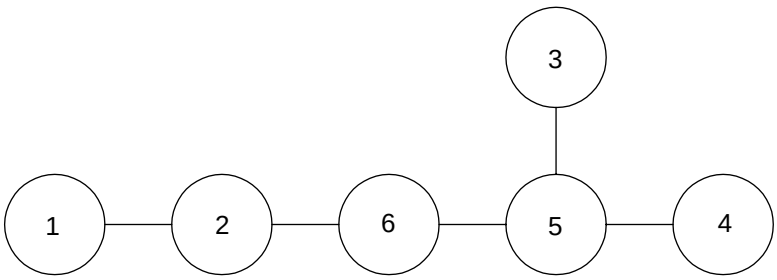

Fig. 5. Min-cut tree T_3.

Link $(3,6)$: Cut $X_1 = \{3\}$ optimal for $\lambda \in (0, 3.5]$ and $X_1 = \{3,5,4\}$ optimal for $\lambda \geqslant 3.5$. Thus $\lambda_1 = 3.5$.

Step 2: We start with cut $X_1 = \{3,5,4\}$ and find a min-cut tree in $G(3.5)$. Following is the min-cut tree T_2 in Fig. 4:

Iteration number 2:

Step 1: We need to determine $G(\lambda)$ for the new links in $T_2 : (3,5)$ and $(3,4)$.

Link $(3,5)$: Cut $X_1 = \{5\}$ optimal for $\lambda \in (0,6]$ and $\bar{X}_1 = \{3\}$ optimal for $\lambda \geqslant 6$.

Link $(3,4)$: Cut $X_1 = \{4\}$ remains optimal for all $\lambda > 0$. Thus $\lambda_2 = 6$.

Step 2: We start with cut $\bar{X}_1 = \{3\}$ and find a min-cut tree in $G(6)$. Following is the min-cut tree T_3 in Fig. 5:

Iteration number 3:

Step 1: No new λ is discovered and the algorithm stops.

Thus, in $G(\lambda)$, T_1 is a min-cut tree for $\lambda \in [0, 3.5], T_2$ for $\lambda \in [3.5, 6]$ and T_3 for $\lambda \geqslant 6$.

4. Applications

The results of this work have applications or strong potential for applications in a variety of areas. In this section we outline a few of these briefly.

4.1. Network flows subject to an arc destruction

Consider an $s-t$ flow and assume that an arc will be destroyed by an adversary. The problem of maximizing the residual flow upon destruction of an arc has been studied recently [2]. Generalization of this to solve for all source–sink pairs efficiently will be facilitated by the results in Section 3 above.

For a given source–sink pair, let $\hat{\lambda}$ be the smallest λ-value for which max flow is obtained. As shown in [2], the arc that will be destroyed has a flow value of $\hat{\lambda}$, and the maximum residual flow value is $v(\hat{\lambda})-\hat{\lambda}$. These values, for all source–sink pairs, can be generated easily from the parametric min-cut trees generated in Section 3. Furthermore, it follows that there are at most $(n-1)$ distinct values of maximal residual flows when we consider the all source–sink pair problem. Also, the set of arcs which will give an arc to be destroyed for each source–sink pair will consist of exactly $(n-1)$ arcs.

4.2. Maximum multiroute flows

In communication networks, improving reliability by using multiroute channels for sending flows has been studied recently [10]. A k-route flow from s to t is defined as a flow where every unit of flow sent by an $s-t$ chain is matched by a unit flow on each of the $(k-1)$ additional $s-t$ arc disjoint chains, so that this unit flow can survive $(k-1)$ arc failures. The problem of maximizing such k-route $s-t$ flows has been addressed and solved in [10] by breaking the problem into two parts: first identifying the maximal value of this flow, and then determining a flow pattern that attains this value. A fairly complicated procedure is given in [10] to find this value. This value, however, can be found trivially by finding the intersection of $v(\lambda)$ function, developed in Section 2, with the line $v=k\lambda$, allowing one to find the maximum k-route flow value for all applicable values of k.

The above can be generalized for finding k-route flows between all pairs of nodes and for all applicable values of k. The analysis of Section 3 will help identify, for each k, the at most $(n-1)$ different values in an efficient manner.

4.3. Flow-based two person games

Consider a two person zero sum game defined as follows. Player 1 sends a flow from a specified source s to a specified sink t. Knowing the flow implemented by Player 1, Player 2 destroys an arc which results in a flow loss. The objective of Player 1 is to maximize this residual flow while that of Player 2 is to minimize the same or equivalently to maximize the flow loss resulting from the destruction of an arc. Thus an optimal strategy for Player 1 is to implement a flow that minimizes the maximum amount of flow on any arc. As discussed in the first application, $\hat{\lambda}$ is the desired maximum flow on any arc. Therefore, an equilibrium strategy for Player 1 is to use maximal flow in $G(\hat{\lambda})$ with $v(\hat{\lambda})=v^*$, and for Player 2 is to destroy any arc with flow $\hat{\lambda}$, resulting in a unique maximal residual flow value of $v(\hat{\lambda})-\hat{\lambda}$. The parametric min-cut trees developed in Section 3 facilitate evaluation of equilibrium solutions for all source–sink pairs.

Acknowledgements

The authors thank the referees for helpful comments on an earlier version of the paper which resulted in improvement of this paper.

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows, Prentice-Hall, Englewood Cliffs, NJ, 1993, Networks 38 (2001) 194–198.
[2] Y.P. Aneja, R. Chandrasekaran, K.P.K. Nair, Maximizing residual flow under an arc destruction, Networks 38 (2001) 194–198.
[3] Y.P. Aneja, K.P.K. Nair, Bicriteria transportation problem, Management Sci. 25 (1979) 73–78.
[4] Y.P. Aneja, K.P.K. Nair, Maximal expected flow in a network subject to arc failures, Networks 10 (1980) 45–57.
[5] R. Chandrasekaran, Minimal ratio spanning trees, Networks 7 (1977) 335–342.
[6] L.R. Ford, D.R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, NJ, 1962.
[7] G. Gallo, M.D. Grigoriadis, R.E. Tarjan, A fast parametric maximum flow algorithm and applications, SIAM J. Comput. 18 (1989) 30–35.
[8] R.E. Gomory, T.C. Hu, Multiterminal network flows, J. SIAM 9 (1961) 551–570.
[9] D. Gusfield. Sensitivity analysis for combinatorial optimization, Ph.D. Thesis, University of California, Berkley, 1980.
[10] W. Kishimoto, A method for obtaining the maximum multiroute flows in networks, Networks 27 (1996) 279–291.
[11] T. Radzik, Newton's method for fractional combinatorial optimization, Proceedings of the 33rd IEEE Symposium on FOCS, 1992, pp. 659–669.