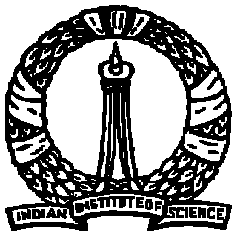
Efficient algorithms for generating all minimum cuts and the cactus representation of a graph

A Project Report
submitted in partial fulfilment of the
requirements for the Degree of
Master of Engineering
in
Faculty of Engineering

by

Arindam Pal



Department of Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012

January 2002

Abstract

In this report we explore some interesting combinatorial properties of minimum cuts of a graph such as submodularity of cuts and crossing cuts lemma. Our main goal is to represent all the (upto $\binom{n}{2}$) min-cuts of a graph compactly, and retrieve them in $O(\lambda)$ time, where λ is the size of any min-cut. We discuss various properties of the cactus, and the algorithm by Karzanov and Timofeev to construct the cactus of an unweighted, undirected graph in $O(\lambda n^2)$ time. Here we propose a new algorithm, which constructs the cactus of an weighted, undirected graph in $O(nm + n^2 \log n + \gamma m \log n)$ time, where γ is the number of cycles in the resulting cactus.

Acknowledgement

I am grateful to my research advisor Prof Ramesh Hariharan for introducing me to the very interesting topic of minimum cuts and cactus representation of a graph. He was a source of constant inspiration for me and always encouraged me to come up with new ideas. He was always an attentive listener to all my results and proofs and pointed out mistakes in some of my arguments. I am also indebted to L. Sunil Chandran for discussing with me various wonderful topics on graph theory and combinatorics. He was always cheerful to come up with new, innovative proofs of some of the problems posed by me.

Contents

Chapter 1

Introduction

The study of minimum cuts is an important part of both graph theory and graph algorithms. These problems have several applications in diverse areas of mathematics and computer science such as VLSI circuit design, network connectivity and reliability problems, study of project networks, combinatorial optimization problems and cutting plane based algorithms for integer programming problems.

This report summarizes the important results and algorithms in this field. Our main intention is to study the combinatorial structure of all the min-cuts of a graph, efficiently generate all of them and to represent them compactly.

1.1 Outline of the report

In Chapter 2, we introduce the notion of cuts and minimum cuts of a graph and discuss the various combinatorial properties, including the submodularity of cuts and the crossing cuts lemma. We also state the circular partition lemma, using which we prove that the number of minimum cuts in a graph is at most $\binom{n}{2}$.

In Chapter 3, we introduce the cactus representation of all min-cuts of a graph and study several properties of the cactus. We also describe the algorithm of Karzanov and Timofeev for constructing the cactus of an unweighted, undirected graph, which takes $O(\lambda n^2)$ time, where λ is the weight of any minimum cut of the graph.

1.2 Our results

In Chapter 4, we will present a new algorithm to construct the cactus representation of an weighted, undirected graph. The novelty of our algorithm is that it decomposes the graph into several subgraphs, computes the cactus of the resulting subgraphs recursively and combines them to form the cactus of the original graph. Thus it is a classic application of the divide and conquer technique. The algorithm constructs the cactus representation of a graph in $O(nm + n^2 \log n + \gamma m \log n)$ time, where γ is the number of cycles in the resulting cactus. In [G93], Gabow has given an algorithm to construct the cactus representation of an weighted, undirected graph in $O(nm \log(n^2/m))$ time. Since, $\gamma = O(n)$ as we have proved, our algorithm is asymptotically as fast as Gabow's algorithm.

Chapter 2

Combinatorial Structure of Minimum Cuts of a Graph

2.1 Definitions and notation

Let $G = (V, E)$ be an undirected graph, where $|V(G)| = n$ and $|E(G)| = m$. For any two sets $X, Y \subseteq V(G)$, we define the following:

$$\Gamma(X, Y) = \{(u, v) \in E(G) : u \in X \text{ and } v \in Y\},$$
$$\Gamma(X) = \Gamma(X, \overline{X}),$$
$$d(X, Y) = |\Gamma(X, Y)|.$$

For undirected graphs $d(X) = d(X, \overline{X})$. For directed graphs $d_{out}(X) = d(X, \overline{X})$ and $d_{in}(X) = d(\overline{X}, X)$

Definition 2.1 Cut A **cut (edge cut)** in a connected graph G is a minimal set of edges whose removal disconnects the graph.

Let $C \subseteq V(G)$ such that $C \neq \emptyset$ and $C \neq V(G)$. The quantity $|\Gamma(C)|$ is called the **capacity** or **weight** or **size** of the cut. The nonempty sets C and $\overline{C}$ are called the **sides** of the cut. Note that the subgraphs induced by the sets C and $\overline{C}$ are connected.

Definition 2.2 Minimum Cut A cut of least capacity is called a **minimum cut** or **min-cut** of the graph G. The capacity of any min-cut is called the **connectivity** or **edge connectivity** of the graph. We use λ to denote the size of the minimum cut.

2.2 Submodularity of cuts

Now we will state and prove an important property called **submodularity of cuts**.

Theorem 2.1 [Submodularity of Cuts] *Let X, Y be two subsets of $V(G)$. Then*

$$d(X) + d(Y) = d(X \cup Y) + d(X \cap Y) + 2d(X - Y, Y - X).$$

Proof Let $A = X - Y, B = \overline{X \cup Y}, C = Y - X$ and $D = X \cap Y$. Then

$$\begin{aligned} d(X) &= d(A, B) + d(A, C) + d(B, D) + d(C, D) \\ d(Y) &= d(A, C) + d(B, C) + d(A, D) + d(B, D) \\ d(X \cup Y) &= d(B) \\ &= d(A, B) + d(B, C) + d(B, D) \\ d(X \cap Y) &= d(D) \\ &= d(A, D) + d(B, D) + d(C, D) \end{aligned}$$

Hence $d(X) + d(Y) - d(X \cup Y) - d(X \cap Y) = 2d(A, C) = 2d(X - Y, Y - X)$,

which is what we wanted to prove.

Corollary 2.1 [Submodular Inequality] *For all $X, Y \subseteq V(G)$:*

$$d(X) + d(Y) \geq d(X \cup Y) + d(X \cap Y).$$

2.3 Crossing cuts and circular partitions

Definition 2.3 Crossing Cuts Two cuts $(X, \overline{X})$ and $(Y, \overline{Y})$ are called **crossing cuts**, if all the four sets $A = X \cap Y, B = X \setminus Y, C = Y \setminus X$, and $D = \overline{X} \cap \overline{Y}$ are non-empty.

Lemma 2.1 *Let $A \neq B$, $A \cap B = \emptyset$ be two minimum cuts such that $T = A \cup B \neq V$ is also a minimum cut. Then $d(A, \overline{T}) = d(B, \overline{T}) = d(A, B) = \lambda/2$.*

Proof Since A, B and T are minimum cuts,

$$d(A, \overline{A}) = d(A, B) + d(A, \overline{T}) = \lambda \quad (1)$$

$$d(B, \overline{B}) = d(A, B) + d(B, \overline{T}) = \lambda \quad (2)$$

$$d(T, \overline{T}) = d(A, \overline{T}) + d(B, \overline{T}) = \lambda \quad (3)$$

From (1) and (2) we have $d(A, \overline{T}) = d(B, \overline{T})$. Combining this with (3) we get $d(A, \overline{T}) = d(B, \overline{T}) = \lambda/2$. Now from (1), $d(A, B) = \lambda/2$.

Lemma 2.2 [Crossing Cuts Lemma] *If X and Y are crossing cuts, then*

1. A, B, C, D are minimum cuts.
2. $d(A, B) = d(B, D) = d(D, C) = d(C, A) = \lambda/2$.
3. $d(A, D) = d(B, C) = 0$.

Proof We know from the submodularity of cuts that

$$d(X) + d(Y) = d(X \cup Y) + d(X \cap Y) + 2d(X - Y, Y - X).$$

Now $\overline{D} = X \cup Y$ and $d(D) = d(\overline{D})$. Hence $d(X) + d(Y) = d(A) + d(D) + 2d(B, C)$. Since X, Y are minimum cuts, $d(X) = d(Y) = \lambda$. Therefore $d(A) + d(D) + 2d(B, C) = 2\lambda$. However $d(A) \geq \lambda$ and $d(D) \geq \lambda$. Thus $d(A) + d(D) + 2d(B, C) \geq 2\lambda$. For equality to hold it must be the case that $d(A) = d(D) = \lambda$, and $d(B, C) = 0$. Similarly we can prove that $d(B) = d(C) = \lambda$, and $d(A, D) = 0$.
Now A and B are two disjoint minimum cuts such that $X = A \cup B \neq V$ is also a minimum cut. By the previous lemma, $d(A, B) = \lambda/2$. Similarly we can prove the other equalities in (2). Hence the lemma follows.

Definition 2.4 Circular Partition A **circular partition** is a partition of V into $k \geq 3$ disjoint subsets $V_1, V_2, ..., V_k$ such that

1. $d(V_i, V_j) = \lambda/2$ when $|i - j| = 1 \mod k$ and zero otherwise.
2. For $1 \leq a < b \leq k, A = \bigcup_{i=a}^{b-1} V_i$ is a minimum cut, and if B is a minimum cut such that neither B nor $\overline{B}$ is of this form, then B or $\overline{B}$ is contained in some V_i.

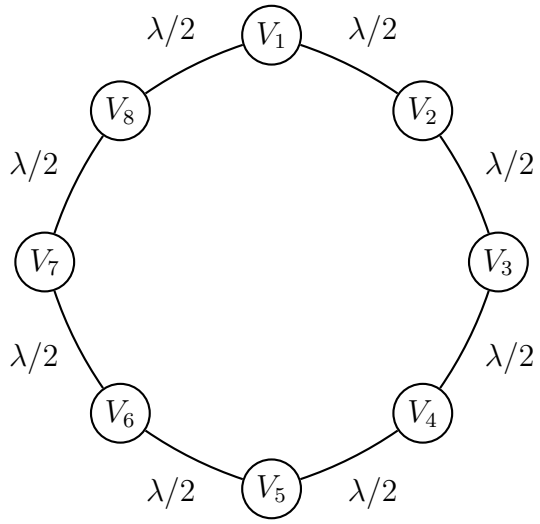


Figure 1: Example of a circular partition

Lemma 2.3 [Circular Partition Lemma] [B75], [DKL76] *If X and Y are two crossing cuts in G, then G has a circular partition $\{V_1, V_2, ..., V_k\}$ such that each of the sets $X \cap Y, X \setminus Y, Y \setminus X$, and $\overline{X} \cap \overline{Y}$ can be represented as $\bigcup_{i=a}^{b-1} V_i$ for appropriate choices of a and b.*

Corollary 2.2 *The number of crossing cuts in a graph is at most $\binom{n}{2} = O(n^2)$, and the number of non-crossing cuts is at most $2n - 1 = O(n)$.*

Proof Let $\mathcal{C}(G)$ be the set of all crossing min-cuts in G. Our goal is to prove that $|\mathcal{C}(G)| \leq \binom{n}{2}$.

The proof is by induction on $|V(G)| = n$.

Let $V_1, V_2, ..., V_k$ be a circular partition in G. If each of the V_i is a single vertex then $k = n$, and G is a cycle on n vertices. Thus any two of the n edges in the cycle can be taken as a min-cut and hence there are $\binom{n}{2}$ min-cuts in G.

However if for some $i, |V_i| > 1$, let $|V_i| = n_i$. Then we have $\sum_{i=1}^{k} n_i = n$ and the number of min-cuts in each V_i is at most $\binom{n_i}{2}$. Moreover, in the cycle containing $V_1, V_2, ..., V_k$ there are exactly $\binom{k}{2}$ min-cuts. By the definition of a circular partition, these are the only min-cuts in G. Hence G has at most $\sum_{i=1}^{k}\binom{n_i}{2} + \binom{k}{2}$ min-cuts.

$$\begin{aligned}
\sum_{i=1}^{k}\binom{n_i}{2} &= \sum_{i=1}^{k} \frac{n_i(n_i-1)}{2} \\
&= \frac{1}{2}\left[\sum_{i=1}^{k} n_i^2 - \sum_{i=1}^{k} n_i\right] \\
&= \frac{1}{2}\left[\left(\sum_{i=1}^{k} n_i\right)^2 - 2\sum_{i<j} n_i n_j - n\right] \\
&\leq \tfrac{1}{2}\left(n^2 - 2\binom{k}{2} - n\right) \quad \text{.....(i)} \\
&= \binom{n}{2} - \binom{k}{2},
\end{aligned}$$

where (i) follows from the fact that $n_i \geq 1$ and there are $\binom{k}{2}$ terms of the form $n_i n_j$.

Therefore $\sum_{i<j} n_i n_j \geq \binom{k}{2}$. Hence $|\mathcal{C}(G)| = \sum_{i=1}^{k}\binom{n_i}{2} + \binom{k}{2} \leq \binom{n}{2}$, and this is what we set out to prove.

The set of non-crossing min-cuts in a graph forms a set of laminar sets. Since a set of laminar sets on n objects can have at most $2n - 1$ elements, it follows that the number of non-crossing min-cuts in a graph is at most $2n - 1 = O(n)$.

It is possible that G has more than one circular partition. If $P = \{V_1, V_2, ..., V_k\}$ and $Q = \{U_1, U_2, ..., U_l\}$ are two distinct circular partitions of V, then these partitions are ***compatible***, if there exists a unique i and j such that $U_p \subseteq V_i,\ \forall p \neq j$ and $V_q \subseteq U_j,\ \forall q \neq i$.

Lemma 2.4 [F99] *Any two circular partitions of a graph are compatible.*

Proof Let P and Q be two distinct circular partitions of V as defined above. For all $1 \leq i \leq k$, there exists a $1 \leq j \leq l$ such that either V_i or $\overline{V_i}$ is contained in U_j. (If not, then $V_i = \bigcup_{j=a}^{b-1} U_j$. Since $X = \bigcup_{j=a+1}^{b} U_j$ is a minimum cut and X and V_i cross each other, it follows that Q does not satisfy the second condition of a circular partition.) Thus, if there is an i such that V_i and V_{i+1} are contained in two different sets of the partition Q, then each $\overline{V_p}, 1 \leq p \leq k, p \neq i, i+1$, is not contained in any U_j. Hence, each $V_p, 1 \leq p \leq k$, is contained in some $U_q, 1 \leq q \leq l$. By the definition of a circular partition, each V_p must be contained in a unique U_q, and hence $P = Q$. If $V_p \subseteq U_j$, for all $1 \leq p \leq k$, then $U_j = V$ and $\overline{U_j} = \emptyset$, contradicting the fact that Q is a circular partition. Hence if $P \neq Q$, there is at least one i and j such that $\overline{V_i} \subseteq U_j$. This means that for all $r \neq i, V_r \subseteq U_j$ and $\overline{U_j} \subseteq V_i$, which is equivalent to saying that $U_s \subseteq V_i$, for $s \neq j$.

Definition 2.5 Laminar Set A set of sets is called **laminar**, if any pair of sets are either disjoint or one set is a subset of the other.

Laminar sets that do not contain the empty set can be represented by a tree, where the root node corresponds to the entire underlying set and the leaves correspond to the sets that contain no other sets. The *parent* of a set A is the smallest set properly containing A. Since the total number of nodes in a tree with n leaves is at most $2n - 1$, the size of the largest set of laminar sets of n objects is at most $2n - 1$. It follows from Lemma 2.4, that the circular partitions of a graph are laminar sets. From this we get the following corollary.

Corollary 2.3 [F99] *There are at most $n - 2$ distinct circular partitions of a graph with n vertices.*

Chapter 3

The Cactus of a Graph and Its Properties

In this chapter, we will define the cactus structure to represent all the min-cuts of a graph. We will study the important properties of the cactus and show how it can be used to design efficient algorithms for min-cut problems.

3.1 The cactus representation

Definition 3.1 CACTUS A **cactus** is a tree-like graph that may contain cycles, but two cycles can have at most one vertex in common. In other words every edge in the cactus lies on at most one cycle. More precisely, a cactus of a graph G denoted by $\mathcal{H}(G)$ has a mapping π that maps sets of vertices of G to vertices of $\mathcal{H}(G)$ and a weight associated with every edge. If λ is the value of the minimum cut in G, then in $\mathcal{H}(G)$ each tree edge has weight λ and each cycle edge has weight $\lambda/2$. The mapping π is such that every minimum cut M of $\mathcal{H}(G)$ corresponds to a minimum cut $\pi^{-1}(M)$ of G, and every minimum cut in G is represented by a minimum cut M of $\mathcal{H}(G)$. The minimum cuts of the cactus are precisely those that cut either one path edge or two cycle edges from the same cycle. If $\pi^{-1}(i) = \emptyset$, we say that i is an **empty node**.

If G has no circular partitions, then G has no crossing cuts. In this case the structure of the cactus of G is very simple. It is a tree.

However if G has crossing cuts, then the cactus must have cycles to represent the circular partitions of G. If G has a circular partition $\{V_1, V_2, ..., V_k\}$, the minimum cuts can be represented by partitions of the form $\bigcup_{i=a}^{b-1} V_i$ for some $1 \leq a < b \leq k$. This can be described by the cuts of a simple cycle, where each V_i corresponds to a node on the cycle.

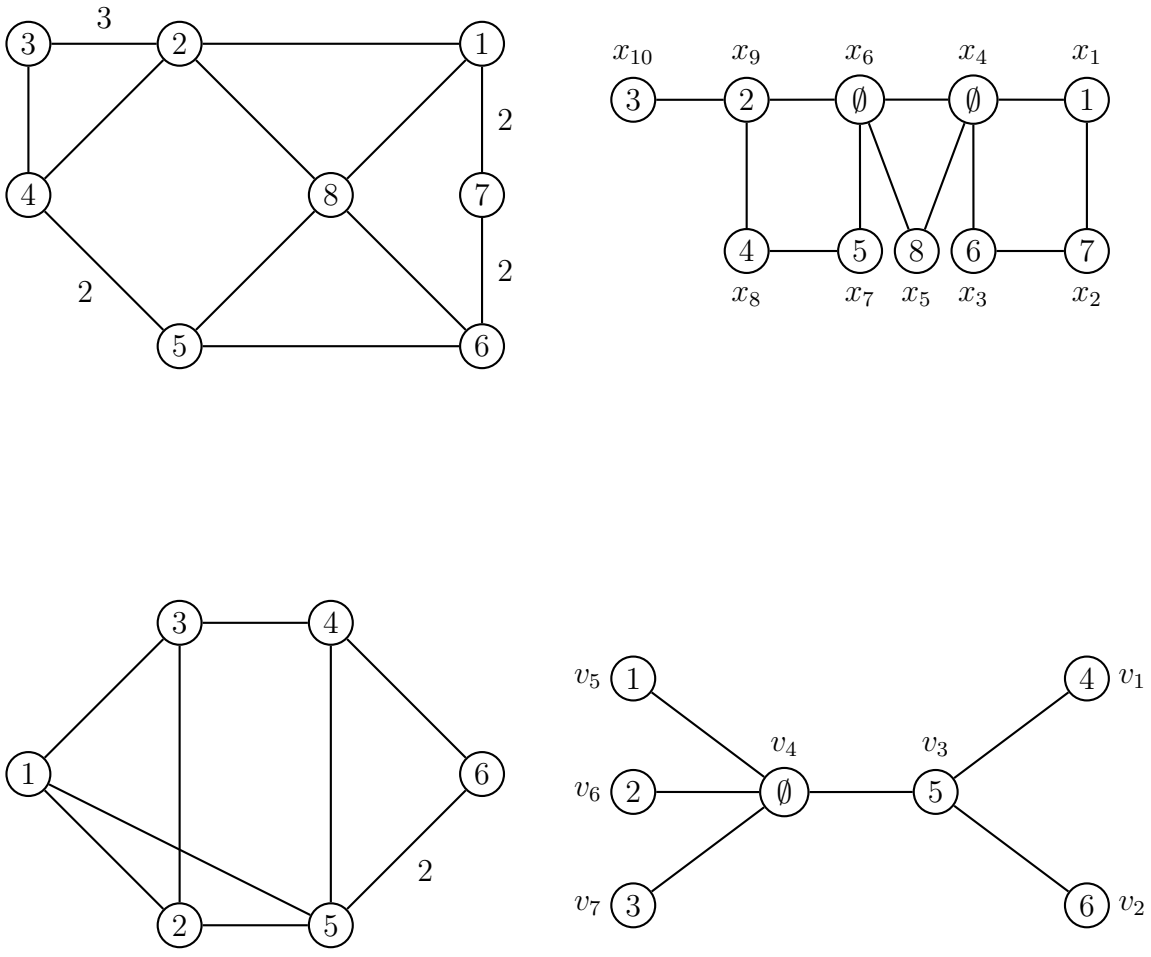


Figure 2: Examples of cactus

3.2 Properties of the cactus

Theorem 3.1 [DKL76], [KT86], [NK94] *Every weighted graph on n vertices has a cactus on at most $2(n-1)$ vertices.*

Lemma 3.1 *A cactus with n vertices can have at most* $\frac{n-1}{2}$ *cycles.*

Proof Let k be the number of cycles in a cactus and let n_i be the number of vertices in the i^{th} cycle. Therefore, $\sum_{i=1}^{k} n_i \leq n + k - 1$, because two cycles can share at most one vertex in common and thus at most $k-1$ vertices can be repeated. But each cycle must have at least three vertices, i.e. $n_i \geq 3$. Hence, $\sum_{i=1}^{k} n_i \geq 3k$. So, $3k \leq n + k - 1$ or $k \leq \frac{n-1}{2}$. Moreover, equality is attained only when $n_i = 3$, for all i. In other words, there can be at most $\frac{n-1}{2}$ cycles and this can happen if and only if all of them are C_3 (triangle).

Lemma 3.2 *A cactus with n vertices can have at most* $\frac{3}{2}(n-1)$ *edges.*

Proof First notice that a cycle on k vertices has k edges, while a path on k vertices has $k-1$ edges. Thus to maximize the number of edges for a fixed number of vertices, we should try to maximize the number of cycles. Hence by the previous lemma, the number of edges is maximized when there are exactly $\frac{n-1}{2}$ triangles in the cactus. Thus a cactus on n vertices can have at most $\frac{3}{2}(n-1)$ edges.

Corollary 3.1 *The cactus of a graph on n vertices can have at most* $n-1$ *cycles.*

Proof By Theorem 3.1, $|V(\mathcal{H}(G))| \leq 2(n-1)$. Let γ be the number of cycles in $\mathcal{H}(G)$. From Lemma 3.1 we have,

$$\gamma \leq \tfrac{1}{2}(|V(\mathcal{H}(G))| - 1) \leq \tfrac{1}{2}(2(n-1) - 1) < n - 1.$$

Corollary 3.2 *The cactus of a graph on n vertices can have at most* $3(n-1)$ *edges.*

Proof By Theorem 3.1, $|V(\mathcal{H}(G))| \leq 2(n-1)$. Hence by virtue of Lemma 3.2,

$$|E(\mathcal{H}(G))| \leq \tfrac{3}{2}(|V(\mathcal{H}(G))| - 1) \leq \tfrac{3}{2}(2(n-1) - 1) < 3(n-1).$$

Now we will describe a simple algorithm to construct the cactus of an unweighted graph. Let $V = \{1, 2, ..., n\}$ be an arbitrary numbering of the vertices of G. For each minimum cut $(A, \overline{A})$ assume that vertex $1 \in A$ and define $j(A)$ to be the smallest indexed vertex that is separated from vertex 1 by $(A, \overline{A})$, i.e. $\{1, 2, ..., j(A) - 1\} \in A$ and $j(A) \in \overline{A}$. For each $i, i = 1, ..., n-1$, define $\mathcal{S}_i = \{(A, \overline{A}) | j(A) = i + 1\}$. Note that the sets $\mathcal{S}_i$ are disjoint and the set of all minimum cuts is $\bigcup_{i=1}^{n-1} \mathcal{S}_i$.

Definition 3.2 CLOSURE **Closure** of a set of vertices A in a directed graph is the smallest set containing A such that no edge is leaving A.

Lemma 3.3 [PQ80] *There is a one-one correspondence between the minimum (s,t) cuts of a graph and the closed vertex sets containing s in the residual graph of a maximum (s,t) flow.*

Suppose that the vertices of G are numbered such that, for any i, vertex $i+1$ is adjacent to at least one of the vertices of the set $V_i = \{1, 2, ..., i\}$. This numbering is known as *adjacency order*. If G is connected, such a representation can always be obtained by a BFS traversal. The following lemma gives some insight on why this scheme makes the representation of $\mathcal{S}_i$ very simple.

Lemma 3.4 *If the vertices of G are numbered in adjacency order, then all cuts in $\mathcal{S}_i$ are non-crossing and hence can be represented by a chain.*

Proof Suppose for the sake of contradiction that $(X, \overline{X})$ and $(Y, \overline{Y})$ are two crossing cuts in $\mathcal{S}_i$ such that $\{1, 2, ..., i\} \subseteq X$ and $\{1, 2, ..., i\} \subseteq Y$. Since $j(X) = j(Y) = i+1, \{1, 2, ..., i\} \subseteq X \cap Y$ and $i+1 \subseteq \overline{X} \cap \overline{Y}$. By the crossing cuts lemma, $d(X \cap Y, \overline{X} \cap \overline{Y}) = 0$. However due to adjacency ordering, there must be at least one edge between $\{1, 2, ..., i\}$ and $i+1$. Thus we get a contradiction and conclude that all cuts in $\mathcal{S}_i$ are non-crossing.
It is now easy to represent a set of non-crossing cuts $(X_j, \overline{X}_j)$ all of which have $\{1, 2, ..., i\} \subseteq X_j$ and $i+1 \subseteq \overline{X}_j$. We just order the sets X_j's by inclusion. This gives a partition of V into $V_1, V_2, ..., V_k$ such that $\{1, 2, ..., i\} \subseteq V_1$ and $i+1 \subseteq V_k$. Any cut $(A, \overline{A})$ where $A = \bigcup_{j=1}^{a} V_j$ (for $a < k$) is a cut in $\mathcal{S}_i$, and any cut in $\mathcal{S}_i$ can be represented in this way.

3.3 The algorithm of Karzanov and Timofeev

Algorithm 3.1 [KT86]

The algorithm of Karzanov and Timofeev for constructing $\mathcal{H}(G)$ is given below:

1. Number the vertices of G in adjacency order.

2. For each $i < n$, compute the maximum flow between $V_i = \{1, 2, ..., i\}$ and $i + 1$. if the flow value is λ, obtain the chain representation C_i of $\mathcal{S}_i$ by computing the strongly connected components of the residual graph. Otherwise C_i is the empty chain.

3. Merge the chains $C_{n-1}, ..., C_1$ into a single cactus $\mathcal{H}(G)$.

Merging Step: Let u be the node in $\mathcal{H}(G^{i+1})$ containing vertices $\{1, 2, ..., i + 1\}$. Merging C_i with $\mathcal{H}(G^{i+1})$ is done in four steps.

1. If $C_i = V_1, V_2, ..., V_k$ then replace u by k new nodes $u_1, u_2, ..., u_k$.

2. For any (tree or cycle) edge (u, w) in $\mathcal{H}(G^{i+1})$, let $W \neq \emptyset$ be the set of vertices in w, or if w is an empty node, the vertices in the first non-empty node $w' \neq u$ reachable from w. Find the subset V_j such that $W \subseteq V_j$ and connect w to u_j.

3. Let U be the set of vertices mapped into u in $\mathcal{H}(G^{i+1})$. Set $\pi(V_j \cap U) = u_j$. All other mappings are unchanged.

4. Delete all empty nodes of degree two. If there is an empty node of degree three with one tree edge and two cycle edge incident on it, contract the tree edge and remove the empty node. Replace all empty nodes of degree three with 3-cycles, i.e. $Y - \Delta$ transformation.

Note that the $Y - \Delta$ transformation in step 4 has to be applied at each step. Otherwise, as the next figure shows, the resulting cactus *may not* represent all the minimum cuts of a graph.

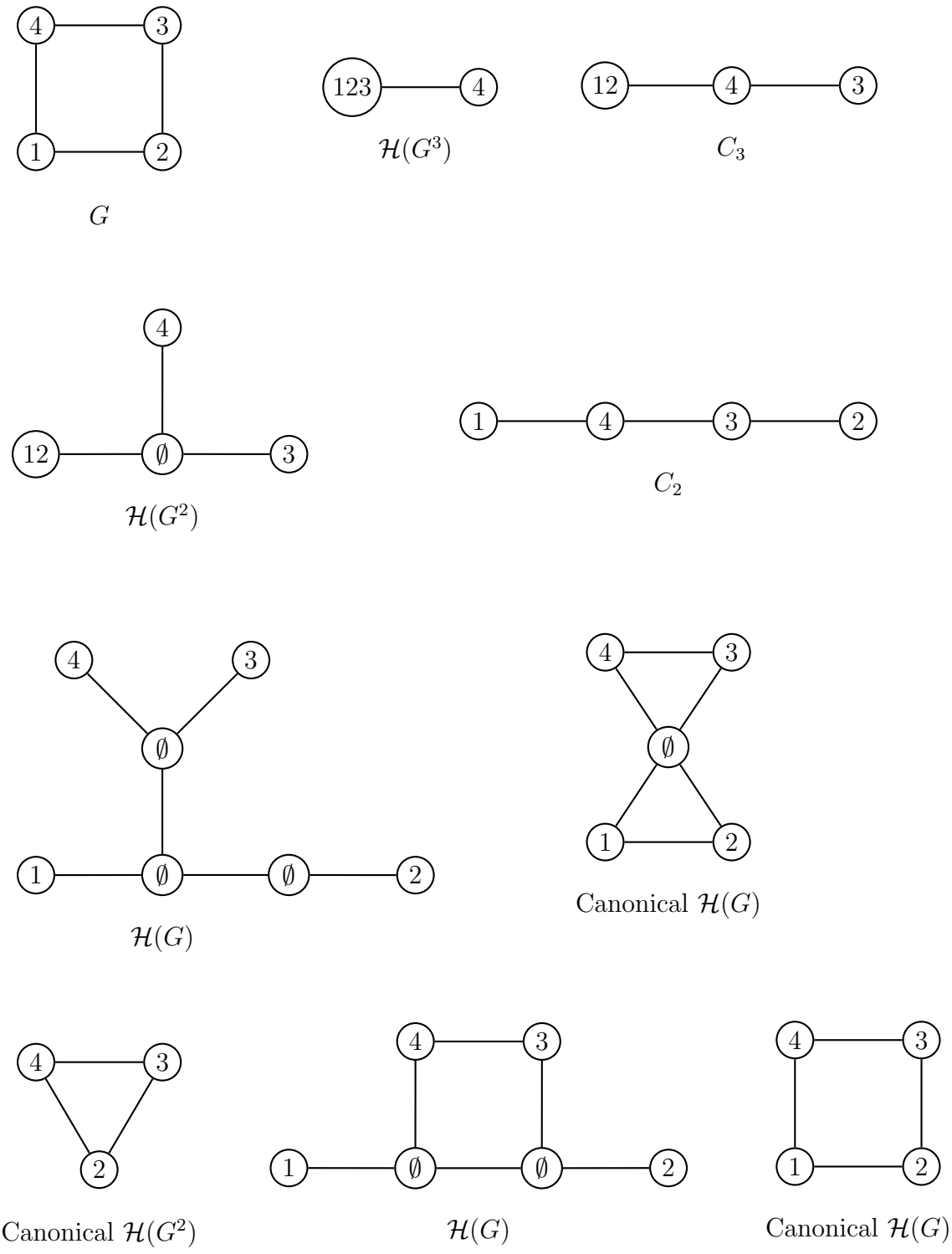


Figure 3: An example to show the importance of applying $Y - \Delta$ transformation at each iteration of the construction of $\mathcal{H}(G)$

We will show in the next two lemmas, that it is always possible to find in step 2, a subset V_j such that $W \subseteq V_j$, and all subtree and cycles that were previously attached to u are now attached to the correct nodes that replace u. Hence, after completing step 3, the resulting graph represents all cuts from $\mathcal{H}(G^{i+1})$ and the cuts in $\mathcal{S}_i$. Since all empty nodes removed in step 4 are redundant, the resulting graph is the desired cactus $\mathcal{H}(G^i)$.

Lemma 3.5 *Let (u, w) be a tree edge in $\mathcal{H}(G^{i+1})$ and $T \subseteq V$ be the set of vertices in the subtree that is attached to u by (u, w). Then $T \subseteq V_j$ for some j.*

Proof Suppose not, and let i be the smallest index such that $T_1 \subseteq V_i$ for some $T_1 \subset T$. This means that there is a min-cut separating $\{1, 2, ..., i\}$ from $i + 1$ and T_1 from $T \setminus T_1$. Also $(T, \overline{T})$ is a min-cut with $\{1, 2, ..., i + 1\} \in \overline{T}$. Since these two are crossing cuts, $T_1, T \setminus T_1, \overline{T}$ form a circular partition and therefore must lie on a cycle of length ≥ 3 in $\mathcal{H}(G^{i+1})$, contradicting the fact that T is attached to u by a tree edge.

Lemma 3.6 *Let (u, w) be a cycle edge in $\mathcal{H}(G^{i+1})$ that lies on a cycle $\mathcal{R}$, and $T_1, T_2, ..., T_r$ be the circular partition induced by $\mathcal{R}$, where $\{1, 2, ..., i + 1\} \subseteq T_1$. Then either*

(i) *$T_2 \cup T_3 \cup ... \cup T_r \subseteq V_j$ for some j, or*

(ii) *There exist a and b such that $T_2 = V_a, T_3 = V_{a+1}, ..., T_r = V_b$.*

Proof Clearly if a cut splits T_1, it cannot split any other T_i $(i \neq 1)$, since such a cut crosses $(T_i, \overline{T}_i)$, so T_1 and the two sides of T_i are in a circular partition. This contradicts the fact that in the partition induced by $\mathcal{R}$, T_i is not split. Hence for any $i > 1, T_i \subseteq V_j$, for some j.

If none of the cuts in $\mathcal{S}_i$ splits $\overline{T}_1$, then $T_2 \cup T_3 \cup ... \cup T_r \subseteq V_j$ for some j. Otherwise, there is a cut that splits both T_1 and $\overline{T}_1$. Since $(T_1, \overline{T}_1)$ is a min-cut, these two are crossing cuts. From the Circular Partition Lemma, there is a circular partition that further refines $\mathcal{R}$ by splitting T_1 into T_1' and T_1''. Hence all cuts $(A, \overline{A})$ where $A = T_1' \cup \bigcup_{k=2}^{i} T_k$ are in $\mathcal{S}_i$, and the result follows.

Theorem 3.2 [KT86] *The Karzanov-Timofeev algorithm constructs a cactus representation of a graph in $O(\lambda n^2)$ time.*

Definition 3.3 For a given subset $\mathcal{C}' \subseteq \mathcal{C}(G)$ of minimum cuts, a pair $(\mathcal{R}, \phi)$ of a cactus $\mathcal{R}$ and a mapping ϕ is called a **cactus representation** for $\mathcal{C}'$ if it satisfies the following conditions:

1. For an arbitrary minimum cut $(S, V(\mathcal{R}) - S) \in \mathcal{C}(\mathcal{R})$, the cut $(X, \overline{X}) \in \mathcal{C}'$, where $X = \{u \in V(G) | \phi(u) \in S\}$ and $\overline{X} = \{v \in V(G) | \phi(v) \in V(\mathcal{R}) - S\}$.
2. Conversely, for any minimum cut $(X, \overline{X}) \in \mathcal{C}'$, there exists a minimum cut $(S, V(\mathcal{R}) - S) \in \mathcal{C}(\mathcal{R})$ such that $X = \{u \in V(G) | \phi(u) \in S\}$ and $\overline{X} = \{v \in V(G) | \phi(v) \in V(\mathcal{R}) - S\}$.

3.4 Union of cactus representations

Suppose that we have two cactus representations $(\mathcal{R}_1, \phi_1)$ and $(\mathcal{R}_2, \phi_2)$ for subsets of minimum cuts $\mathcal{C}_1, \mathcal{C}_2 \subseteq \mathcal{C}(G)$. If there exists nodes $z_1 \in V(\mathcal{R}_1)$ and $z_2 \in V(\mathcal{R}_2)$ such that

$$\phi_1^{-1}(z_1) \cup \phi_2^{-1}(z_2) = V(G),$$

then it is possible to combine $(\mathcal{R}_1, \phi_1)$ and $(\mathcal{R}_2, \phi_2)$ into a single cactus representation $(\mathcal{R}, \phi)$ for $\mathcal{C}_1 \cup \mathcal{C}_2$. Such a cactus $\mathcal{R}$ can be obtained by identifying z_1 and z_2 into a single node z, and keeping all other nodes unchanged. z is a *joint node* in $\mathcal{R}$. The mapping $\phi : \{V(G), \emptyset\} \mapsto V(\mathcal{R}_1) \cup V(\mathcal{R}_2) \cup \{z\} - \{z_1, z_2\}$ is defined as follows:

$$\begin{aligned}
&\phi^{-1}(z) := \phi_1^{-1}(z_1) \cap \phi_2^{-1}(z_2), \\
&\phi^{-1}(x) := \phi_1^{-1}(x) \ \forall x \in V(\mathcal{R}_1) - \{z_1\}, \\
&\phi^{-1}(y) := \phi_2^{-1}(y) \ \forall y \in V(\mathcal{R}_2) - \{z_2\}.
\end{aligned}$$

We denote this operation by $(\mathcal{R}, \phi) := (\mathcal{R}_1, \phi_1) \oplus (\mathcal{R}_2, \phi_2)$. $(\mathcal{R}, \phi)$ is called the **union** of the cactus representations $(\mathcal{R}_1, \phi_1)$ and $(\mathcal{R}_2, \phi_2)$.
Note that the resulting $(\mathcal{R}, \phi)$ may not be a CNCR for $\mathcal{C}_1 \cup \mathcal{C}_2$, even though both $(\mathcal{R}_1, \phi_1)$ and $(\mathcal{R}_2, \phi_2)$ are CNCRs. But $(\mathcal{R}, \phi)$ can be simplified to a CNCR by applying a constant number of 2-cycle contractions and 3-cycle insertions. However, if the joint node z is a 3-junction empty node, then as the following lemma shows, $(\mathcal{R}, \phi)$ can be simplified to a CNCR by applying at most four such operations.

Lemma 3.7 *Let z be the joint node in the union operation of two CNCRs $(\mathcal{R}_1, \phi_1)$ and $(\mathcal{R}_2, \phi_2)$. If z is a 3-junction empty node in $(\mathcal{R}, \phi)$, then $(\mathcal{R}, \phi)$ can be simplified to a CNCR by applying one 3-cycle insertion at z, followed by at most three 2-cycle contractions.*

Proof We first apply a 3-cycle insertion at z. Let y_1, y_2 and y_3 be the new empty nodes. Clearly, the new cactus has no 3-junction empty node, since $(\mathcal{R}_1, \phi_1)$ and $(\mathcal{R}_2, \phi_2)$ are CNCRs. Since the 3-cycle insertion introduces a new cycle (y_1, y_2, y_3), some of the y_i's may be 2-junction empty nodes belonging to a 2-cycle. These nodes can be eliminated by applying at most three 2-cycle contractions. This creates no new 3-junction empty node in $(\mathcal{R}, \phi)$. Hence, the resulting cactus is a CNCR.

3.5 (s,t)-MC-partition

Given an ordered partition $(V_1, ..., V_r)$ and two indices h and k $(1 \leq h \leq k \leq r)$, we define

$$V_{(h,k)} \equiv V_h \cup V_{h+1} \cup ... \cup V_k.$$

For a subset $\mathcal{C}' \subseteq \mathcal{C}(G)$, an ordered partition $(V_1, ..., V_r)$ of $V(G)$ is called a *minimum cut partition* (*MC-partition*) over $\mathcal{C}'$, if

$$\{(V_{(1,k)}, \overline{V_{(1,k)}}) | 1 \leq k \leq r-1\} \subseteq \mathcal{C}'.$$

An edge $e = (s,t)$ is *critical* in G, if $\lambda_G(s,t) = \lambda(G)$. Hence if we remove a critical edge, the edge connectivity of the graph decreases. Let $\mathcal{C}_{(s,t)}(G)$ be the set of all minimum cuts in $\mathcal{C}(G)$ that separates s and t.

Lemma 3.8 *For a critical edge $(s,t) \in E(G)$, all minimum cuts in $\mathcal{C}_{(s,t)}(G)$ can be represented by an MC-partition over $\mathcal{C}(G)$.*

Proof We first show that no two cuts in $\mathcal{C}_{(s,t)}(G)$ cross each other. If two cuts $(X, \overline{X})$ and $(Y, \overline{Y})$ cross each other, then $c_G(X \cap Y, \overline{X \cup Y}) = 0$, by the crossing cuts lemma. But this contradicts the fact that $(s,t) \in E(X \cap Y, \overline{X \cup Y})$. Hence, there are no crossing cuts in $\mathcal{C}_{(s,t)}(G)$. Since any cut in $\mathcal{C}_{(s,t)}(G)$ separates s and t, all minimum cuts in

$\mathcal{C}_{(s,t)}(G)$ can be represented as $(X_1, \overline{X_1}), (X_2, \overline{X_2}), ..., (X_{r-1}, \overline{X_{r-1}})$ such that $s \in X_i \subset X_j$ holds for all $i < j$. Thus the ordered partition $(V_1, V_2, ..., V_r)$ such that
$V_1 = X_1, V_r = \overline{X_{r-1}}$ and $V_i = X_i - X_{i-1}$ for $i = 2, ..., r-1$ satisfies
$\{(V_{(1,k)}, \overline{V_{(1,k)}}) | 1 \leq k \leq r-1\} = \mathcal{C}_{(s,t)}(G)$.

An MC-partition as in the above lemma is called an (s,t)-*MC-partition* over $\mathcal{C}(G)$ and is denoted by $\pi_{(s,t)}$. It is known ([KT86], [NV91]) that an (s,t)-*MC-partition* can be computed in $O(m+n)$ time from an arbitrary flow $F_{(s,t)}$ between s and t. Note that $\pi_{(s,t)}$ can be regarded as a cactus representation $(\mathcal{R}, \phi)$ for $\mathcal{C}_{(s,t)}(G)$.

3.6 (s,t)-cactus representation

Definition 3.4 A cut $(X, \overline{X})$ **crosses** an ordered partition $(V_1, ..., V_r)$ of $V(G)$ if $(X, \overline{X})$ crosses some cut $(V_i, \overline{V_i})$.

Lemma 3.9 [NK96] *Let (s,t) be a critical edge in a graph G. Then no minimum cut $(X, \overline{X})$ crosses the (s,t)-MC-partition $\pi_{(s,t)}$ over $\mathcal{C}(G)$.*

Definition 3.5 A cut $(X, \overline{X})$ is **compatible** with an ordered partition $(V_1, ..., V_r)$ of $V(G)$, if

$$V_i \subseteq X \text{ or } V_i \subseteq \overline{X} \text{ for all } i = 1, 2, ..., r.$$

Let $\mathcal{C}_{comp}(\pi)$ be the set of all minimum cuts in $\mathcal{C}(G)$ that are compatible with π. For an (s,t)-MC-partition $\pi_{(s,t)} = (V_1, ..., V_r)$, any minimum cut $(X, \overline{X})$ satisfies either $(X, \overline{X}) \in \mathcal{C}_{comp}(\pi_{(s,t)})$ or $X \subset V_i$ for some i.
The next lemma shows how to compute all minimum cuts in $\mathcal{C}_{comp}(\pi_{(s,t)})$.

Lemma 3.10 [NK96] *For a critical edge (s,t) in a graph G, let $\pi_{(s,t)} = (V_1, ..., V_r)$ be an (s,t)-MC-partition over $\mathcal{C}(G)$. Then any minimum cut $(X, \overline{X}) \in \mathcal{C}_{comp}(\pi_{(s,t)})$ satisfies the following:*
(i) *If $(X, \overline{X})$ separates s and t, then either $X = V_{(i,r)}$ or $\overline{X} = V_{(i,r)}$ for some i $(1 < i \leq r)$.*
(ii) *If $(X, \overline{X})$ does not separate s and t, then either $X = V_{(i,j)}$ or $\overline{X} = V_{(i,j)}$ for some i, j $(1 < i \leq j < r)$ such that $(V_k, \overline{V_k}) \in \mathcal{C}(G)$ for all k $(i \leq k \leq j)$.*

Any (s,t)-MC-partition $\pi_{(s,t)}$ for a critical edge (s,t) have a cactus representation for $\mathcal{C}_{comp}(\pi_{(s,t)}) \subseteq \mathcal{C}(G)$. This is called an (s,t)-*cactus representation* and denoted by $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$. It is known ([NK96]), that such a representation exists.

Lemma 3.11 [NK96] *Given an (s,t)-MC-partition $\pi_{(s,t)}$ for a critical edge (s,t) in G, a cycle type normal (s,t)-cactus representation $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ can be constructed in $O(m+n)$ time.*

Now we are ready to give an efficient algorithm to construct the cactus representation for an weighted, undirected graph.

Chapter 4

A New Algorithm for Constructing the Cactus

4.1 Decomposition of a graph into subgraphs

Let (s,t) be a critical edge in a graph G. Since no minimum cut in $\mathcal{C}(G) - \mathcal{C}_{comp}(\pi_{(s,t)})$ crosses the (s,t)-MC-partition $\pi_{(s,t)}$, we have the following lemma.

Lemma 4.1 *Let $\pi_{(s,t)} = (V_1, ..., V_r)$ be the (s,t)-MC-partition for a critical edge (s,t) in a graph G. Then for each minimum cut $(X, \overline{X}) \in \mathcal{C}(G) - \mathcal{C}_{comp}(\pi_{(s,t)})$, there is a subset V_i such that*

$$X \subset V_i \text{ or } \overline{X} \subset V_i,$$

and some two vertices $x, y \in V_i$ are separated by $(X, \overline{X})$.

For an (s,t)-MC-partition $\pi_{(s,t)} = (V_1, ..., V_r)$, let $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ be the (s,t)-cactus representation, and G_i be the graph constructed from G by contracting $V(G) - V_i$, $i = 1, 2, .., r$, into a single vertex $\overline{v_i}$. Clearly, $\lambda(G_i) \geq \lambda(G)$. The graph G_i is called **critical** if $\lambda(G_i) = \lambda(G)$. Suppose that G_i is critical and consider a minimum cut $(Y, V(G_i) - Y) \in \mathcal{C}(G_i)$. By Lemma 4.1, either

1. $(Y, V(G_i) - Y)$ separates some two vertices x and y in V_i and satisfies $Y \subseteq V_i$ or $V(G_i) - Y \subseteq V_i$, or
2. $(Y, V(G_i) - Y) = (V_i, \overline{v_i})$.

Note that the cut $(V_i, \overline{v_i})$ is also represented in $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ as $(V_i, V(G) - V_i)$. The ordered tuple $(G_1, ..., G_r)$ is called an (s,t)-*decomposition* of G. For every critical graph G_i, we construct a cactus representation $(\mathcal{R}_{G_i}, \phi_{G_i})$ for $\mathcal{C}(G_i) - (V_i, \overline{v_i})$. Thus every minimum cut in G is represented in $(\mathcal{R}_{G_i}, \phi_{G_i})$ or $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$.
Now suppose that, for $i = 1, ..., r$ the vertex $\overline{v_i}$ in G_i is mapped to z_i in $\mathcal{R}_{G_i}$, i.e. $\overline{v_i} \in \phi_{G_i}^{-1}(z_i)$, and let v_i be the vertex in $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ such that $\phi_{(s,t)}^{-1}(v_i) = V_i$. We restore ϕ_{G_i} by replacing $\overline{v_i}$ with $V(G) - V_i$. It follows that,

$$\phi_{(s,t)}^{-1}(v_i) \cup \phi_{G_i}^{-1}(z_i) = V(G).$$

Hence, we can combine $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ with $(\mathcal{R}_{G_i}, \phi_{G_i})$ by performing the union operation. So, we combine all the cactus representations as follows:
Initialize $(\mathcal{R}, \phi) := (\mathcal{R}_{(s,t)}, \phi_{(s,t)})$. Then, for $i = 1, ..., r$ do $(\mathcal{R}, \phi) := (\mathcal{R}, \phi) \oplus (\mathcal{R}_{G_i}, \phi_{G_i})$ (after modifying ϕ_{G_i} as shown above).

4.2 Decomposition tree of a graph

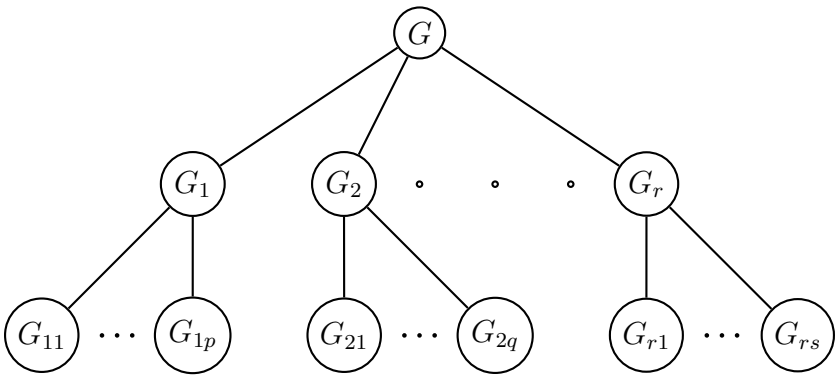


Figure 4: Decomposition tree of a graph

Let G be a graph and let $G_1, G_2, ..., G_r$ be the graphs obtained by an (s,t)-decomposition of G. The rooted tree shown in figure 4 is called the *decomposition tree* of G. Eventually, the given graph G is decomposed into graphs with one or two vertices. If a graph G' with two vertices u and v is generated and its edge (u, v) is critical (i.e. $\lambda_{G'}(u, v) = \lambda$), then the algorithm will decompose G' into two graphs G'_1

and G'_2 which are identical to G'. Hence, to avoid such a trivial decomposition, we check whether the resulting (s,t)-cactus representation contains any new minimum cut. For the current graph G' in the decomposition tree, a minimum cut is called *new*, if it is not found in any graph G'' which is a *proper ancestor* of G'. Otherwise, it is called *old*. If the current (s,t)-cactus representation represents no new minimum cut, we contract s and t into a single vertex and apply the procedure CACTUS recursively to the resulting graph. With this modification, we don't have to duplicate effort for minimum cuts which have already been represented in the cactus of some other graphs.

4.3 Checking for old cuts

Let $\pi_{(s,t)} = (V_1, ..., V_r)$ be an (s,t)-MC-partition and $(G_1, ..., G_r)$ be its (s,t)-decomposition of G. By Lemma 3.8, no two G_i and G_j contain the same minimum cut in $\mathcal{C}(G)$. Thus a minimum cut $(X, \overline{X}) \in \mathcal{C}_{comp}(\pi_{(s,t)})$ is in $\mathcal{C}(G_i)$ only if $X = V_i$ for $V(G_i) = V_i \cup \{\overline{v_i}\}$. Hence, such a minimum cut $(V_i, \overline{v_i})$ separates a single vertex $\overline{v_i}$ from rest of the vertices. For a graph G in the (s,t)-decomposition, let $\mathcal{C}_{old}(G)$ be the set of old minimum cuts in G. By induction, we see that any old minimum cut in the current graph G separates a single vertex w from $V(G) - w$. We maintain $\mathcal{C}_{old}(G)$ as $\{w_1, ..., w_k\}$ to imply that $(w_i, V(G) - w_i) \in \mathcal{C}_{old}(G)$, for $i = 1, ..., k$.

We first show how to compute $\mathcal{C}_{old}(G_i)$ for each graph G_i in an (s,t)-decomposition $(G_1, ..., G_r)$ of G. Suppose that we are given a set $\mathcal{C}_{old}(G) = \{w_1, ..., w_k\}$ of old minimum cuts in G and let $V(G_i) = V_i \cup \{\overline{v_i}\}$, where $\overline{v_i}$ is the vertex contracted from $V(G) - V_i$. Obviously, a minimum cut $\{w_j, V(G) - w_j\}$ will be preserved in the graph G_i such that $w_j \in V_i$. The set of all such vertices is given by $\mathcal{C}_{old}(G) \cap V_i$. For each G_i, if $(V_i, V(G) - V_i) \in \mathcal{C}(G)$, then such a minimum cut $(V_i, \overline{v_i})$ is also kept in G_i. Hence, $\mathcal{C}_{old}(G_i)$ can be computed from $\mathcal{C}_{old}(G)$ as shown below:

$$\mathcal{C}_{old}(G_i) := (\mathcal{C}_{old}(G) \cap V_i) \cup \{\overline{v_i} | c_G(V_i, \overline{v_i}) = \lambda\},$$

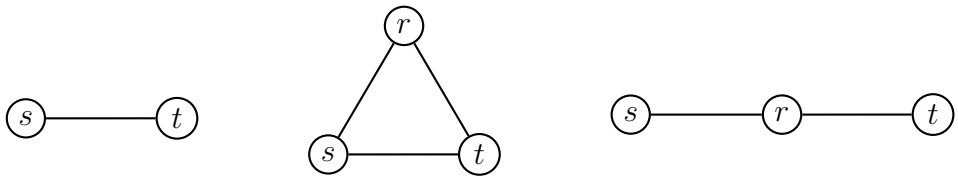
where $\lambda = \lambda(G)$ is the size of a minimum cut in G.

Next we consider how to check whether an (s,t)-cactus representation for the current graph G contains only old minimum cuts in $\mathcal{C}_{old}(G)$. For this we need the following

result.

Lemma 4.2 *Let $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ be a cycle-type normal (s,t)-cactus representation for a critical edge (s,t) in a graph G, and let r be the number of non-empty nodes in $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$. Then*

(i) *If $r \leq 3$, then $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ is equivalent to one of the (s,t)-cactus representations shown in* Figure 5.
(ii) *If $r \geq 4$, then $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ represents at least one minimum cut $(X, \overline{X}) \in \mathcal{C}(G)$ such that both $\phi_{(s,t)}(X)$ and $\phi_{(s,t)}(\overline{X})$ contain more than one non-empty node.*



(a) (b) (c)

Figure 5: (a) The cycle-type normal (s,t)-cactus representation containing two non-empty nodes ($s \in V_1, t \in V_2$). (b), (c) The cycle-type normal (s,t)-cactus representations containing three non-empty nodes ($s \in V_1, r \in V_2, t \in V_3$).

Proof Let $(V_1, ..., V_r)$ be an (s,t)-MC-partition over $\mathcal{C}(G)$, where $s \in V_1$, $t \in V_r$, and r gives the number of non-empty nodes in $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$, where $r \geq 2$. Obviously, every 1-junction node is a non-empty node in a cactus representation.
(i) We first show that, if $r \leq 3$, then $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ has no empty node. Assume that there is an empty node z, and let $\mathcal{R}_{(s,t)} - z$ denote the graph obtained from the cactus $\mathcal{R}_{(s,t)}$ by removing the node z together with all the incident edges. Since $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ is a CNCR, z cannot be a 2-junction or 3-junction empty node. If z is a k-junction empty node for some $k \geq 4$, then each of the k components in $\mathcal{R}_{(s,t)} - z$ contains a 1-junction node which are non-empty nodes, implying $r \geq 4$, a contradiction. Hence, $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ has no empty node. Thus it is easy to see that it has to be equivalent to any one of the representations given in Figure 5.

(ii) If $r \geq 4$, then a cut $(V_{(1,2)}, V_{(3,r)})$ is a minimum cut such that each of $\phi_{(s,t)}(V_{(1,2)})$ and $\phi_{(s,t)}(V_{(3,r)})$ contains at least two non-empty nodes.

Lemma 4.3 *For a graph G' in the decomposition tree of a graph G, let $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ be a cycle-type normal (s,t)-cactus representation for an edge $(s,t) \in E(G')$ with $\lambda_{G'}(s,t) = \lambda(G)$. Then $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ represents no new minimum cut only if it is equivalent to one of the three representations shown in* Figure 5.

Proof If $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ has at least four non-empty nodes, then by Lemma 4.2, there is a minimum cut $(X, V(G') - X)$ such that both $\phi_{(s,t)}(X)$ and $\phi_{(s,t)}(V(G') - X)$ has at least two non-empty nodes. Such a cut is not in $\mathcal{C}_{old}(G')$, since every cut in $\mathcal{C}_{old}(G')$ separates a single node w from rest of the vertices $V(G') - w$. Therefore, if $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ represents no new minimum cut, it must have at most three non-empty nodes, and by Lemma 4.2 it must be equivalent to one of the three representations shown in Figure 5.

From the above lemmas, we can easily check whether the cycle-type normal (s,t)-cactus representation for the current graph G' represents a new minimum cut. If the (s,t)-cactus representation has at least four non-empty nodes, then it represents at least one new minimum cut. Otherwise, it represents at most three minimum cuts. We can check if each of these cuts is old or new by testing whether it belongs to $\mathcal{C}_{old}(G')$. If the (s,t)-cactus representation contains no new minimum cut, then $V_1 = \{s\}$ or $V_r = \{t\}$ holds for $s \in V_1$ and $t \in V_r$ in the (s,t)-MC-partition $(V_1, ..., V_r)$ of $V(G')$. In this case, we contract s and t into a single vertex and update the set of old minimum cuts by $\mathcal{C}_{old}(G'') = \mathcal{C}_{old}(G') - \{s,t\}$ for the graph G'' obtained from the contraction of s and t. The number of non-empty nodes r, in the (s,t)-cactus representation can be obtained in $O(n)$ time. In the case when $r \leq 3$, we can check if each of at most three minimum cuts belongs to $\mathcal{C}_{old}(G')$ in $O(|\mathcal{C}_{old}(G')|) = O(n)$ time. Contracting two vertices s and t and updating $\mathcal{C}_{old}(G')$ can also be done in $O(n)$ time.

4.4 Informal description of the algorithm

In this section, we give the basic ideas behind the algorithm. The set $\mathcal{C}_{old}(G)$ is the set of old minimum cuts in G. Any minimum cut in $\mathcal{C}_{old}(G)$ separates a single vertex v from

$V(G) - v$. There can be two types of minimum cuts in G.

(i) Minimum cuts separating s and t.
(ii) Minimum cuts of the form $(X, \overline{X})$ which don't separate s and t, and for which, either $X \subset V_i$ or $\overline{X} \subset V_i$ for some V_i.

We pick an arbitrary edge (s,t) in G and compute $\lambda_G(s,t)$. If (s,t) is not a critical edge (i.e. $\lambda_G(s,t) < \lambda(G)$), then the edge is contracted. Otherwise, we construct the (s,t)-MC-partition and the (s,t)-cactus representation $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ for the edge (s,t). If all minimum cuts represented by $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ are old cuts (i.e. in $\mathcal{C}_{old}(G)$), then we contract the edge (s,t) and remove the vertices s and t from $\mathcal{C}_{old}(G)$. Otherwise, for each i, we contract the vertices of $V(G) - V_i$ into a single vertex $\overline{v_i}$ and denote the resulting graph as G_i. Any minimum cut of G of type (i) will be present in $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ for G, and any minimum cut of G of type (ii) will be present in one of the cacti $(\mathcal{R}_{G_i}, \phi_{G_i})$ for some subgraph G_i. If we take the union of these cacti, we will get all the minimum cuts of G, and also its cactus representation $(\mathcal{R}, \phi)$.

4.5 The algorithm

In this section we will give an efficient algorithm to construct the cactus of an undirected graph.

Algorithm 4.1 CONSTRUCT(G)

Input: An undirected graph G.
Output: A CNCR $(\mathcal{R}, \phi)$ for $\mathcal{C}(G)$.
Compute $\lambda(G)$;
$\mathcal{C}_{old}(G) := \emptyset$;
$(\mathcal{R}, \phi) :=$ CACTUS$(G, \mathcal{C}_{old}(G), \lambda(G))$.

The procedure CACTUS$(G, \mathcal{C}_{old}(G), \lambda)$ which constructs the CNCR $(\mathcal{R}, \phi)$ is given below.

Note: In the following discussion we denote by $G|X$, the graph obtained by contracting all the vertices of $X \subseteq V(G)$ into a single vertex.

Procedure CACTUS$(G, \mathcal{C}_{old}(G), \lambda)$

Input: An undirected graph G, a set $\mathcal{C}_{old}(G)$ of vertices and a positive real number λ.

Output: A cactus representation $(\mathcal{R}, \phi)$ for a class $\mathcal{C}'$ of min-cuts such that $\mathcal{C}(G) - \mathcal{C}_{old}(G) \subseteq \mathcal{C}' \subseteq \mathcal{C}(G)$.

while $|V(G)| > 1$ **do** 1
  choose an edge (s,t) and compute $\lambda_G(s,t)$; 2
  if $\lambda_G(s,t) > \lambda$ **then** 3
    $G := G|\{s,t\}$; 4
  else /* $\lambda_G(s,t) = \lambda$ */ 5
    compute a maximum flow $F_{(s,t)}$ between s and t in G; 6
    construct the (s,t)-MC-partition $\pi_{(s,t)} = (V_1, ..., V_r)$ and the 7
    cycle-type normal (s,t)-cactus representation $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ from $F_{(s,t)}$;
    if all minimum cuts represented by $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ are in $\mathcal{C}_{old}(G)$ **then** 8
      $G := G|\{s,t\}$; $\mathcal{C}_{old}(G) := \mathcal{C}_{old}(G) - \{s,t\}$; 9
    else 10
      for each non-empty node $v_i \in \mathcal{R}_{(s,t)}$ **do** 11
        $G_i := G|(V(G) - V_i)$; /* contract $V(G) - V_i$ into a single vertex $\overline{v}_i$ */ 12
        $\mathcal{C}_{old}(G_i) := (\mathcal{C}_{old}(G) \cap V_i) \cup \{\overline{v}_i | c_G(V_i, \overline{v}_i) = \lambda\}$; 13
      end /* for */ 14
      $(\mathcal{R}, \phi) := (\mathcal{R}_{(s,t)}, \phi_{(s,t)}) \oplus$ CACTUS$(G_1, \mathcal{C}_{old}(G_1), \lambda) \oplus ...$ 15
      $\oplus$ CACTUS$(G_r, \mathcal{C}_{old}(G_r), \lambda)$, after replacing $\overline{v}_i$ with $V(G) - V_i$ in each CACTUS$(G_i, \mathcal{C}_{old}(G_i), \lambda)$;
      convert $(\mathcal{R}, \phi)$ into a CNCR; 16
      return $(\mathcal{R}, \phi)$; 17
    end /* if */ 18
  end /* if */ 19
end /* while */ 20
return the trivial cactus $(\mathcal{R}, \phi)$. 21

4.6 Analysis of the algorithm

4.6.1 Correctness

First note that s and t are contracted in procedure CACTUS only when there is no new minimum cut separating s and t in the current graph G. In decomposing the current graph G into $G_1, G_2, ..., G_r$ in the **for** loop of CACTUS, any minimum cut in $\mathcal{C}(G) - \mathcal{C}_{comp}(\pi_{(s,t)})$ is present in at least one of the new graphs, while any minimum cut in $\mathcal{C}_{comp}(\pi_{(s,t)})$ is present in at most one of the graphs. Hence, during the execution of CONSTRUCT, neither edge contraction nor graph decomposition destroys any minimum cut in G. Moreover the decomposition tree associated with the execution of CONSTRUCT is finite, since detecting old minimum cuts in CACTUS prevents the same graph from appearing in the decomposition tree more than once. Therefore CONSTRUCT correctly computes a cactus representation for $\mathcal{C}(G)$ of a graph G.

4.6.2 Time complexity

In this subsection, we analyze the running time of the algorithm CONSTRUCT. Computing the edge-connectivity $\lambda(G)$ of a graph G in CONSTRUCT can be done in $O(nm + n^2 \log n)$ time by using the algorithm in [NI92]. First, we show how to compute $\lambda_G(s,t)$ and a maximum flow $F_{(s,t)}$ in CACTUS. For this we need the following definition:

Definition 4.1 A **maximum adjacency ordering (MA-ordering)** is a total ordering $v_1, v_2, ..., v_n$ of the vertices in $V(G)$ such that

$$c_G(\{v_1, v_2, ..., v_i\}, v_{i+1}) = \max_{u \in \{v_{i+1}, ..., v_n\}} [c_G(\{v_1, v_2, ..., v_i\}, u)] \qquad (1 \leq i \leq n-1).$$

Lemma 4.4 [NI92] *For a graph G with n vertices and m edges, an MA-ordering can be found in $O(m + n \log n)$ time, and $\lambda_G(v_{n-1}, v_n) = c_G(v_n)$ holds for the last two vertices v_{n-1} and v_n. Moreover, a maximum flow between the last two vertices v_{n-1} and v_n can be computed in $O(m \log n)$ time.*

Hence, by choosing the last two vertices in an MA-ordering of G as the vertices s and t, $\lambda_G(s,t)$ can be computed in $O(m + n \log n)$ time, and a maximum flow between s and t can be computed in $O(m \log n)$ time. The **for** loop can be executed in $O(n)$ time.

The (s,t)-MC-partition $\pi_{(s,t)}$ and the (s,t)-cactus representation $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ can also be computed in $O(n)$ time. Now there are two cases to consider:
(a) $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ represents no new minimum cuts, so s and t are contracted.
(b) $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ represents at least one new minimum cut, so the current graph G is decomposed into $G_1, ..., G_r$ in the **for** loop in CACTUS.
Let γ be the number of cycles in a CNCR for $\mathcal{C}(G)$ of a graph G. The next lemma shows that case (b) can occur at most $O(\gamma)$ times during the execution of CONSTRUCT.

Lemma 4.5 *During the execution of algorithm* CONSTRUCT, case (b) of CACTUS *can occur at most* 4γ *times.*

Proof Let us consider the decomposition tree of CONSTRUCT for an input graph G'. For a cactus representation $(\mathcal{R}', \phi')$ of the current graph G' in the decomposition tree, we say that a cycle C in the cactus *represents* a minimum cut $(X, V(G') - X)$ in G', if removal of some two edges in C creates two components S and $V(\mathcal{R}') - S$ such that $\phi^{-1}(S) = X$ and $\phi^{-1}(V(\mathcal{R}') - S) = V(G') - X$. A cycle C in $\mathcal{R}'$ is called *proper*, if it represents at least one new minimum cut in G'. Note that, if case (b) occurs, then the (s,t)-cactus representation in line 7 must have a cycle $C_{(s,t)}$ representing at least one new minimum cut. We call $C_{(s,t)}$ a *new cycle* with respect to the (s,t)-cactus representation. Let c^* denote the total number of new cycles over all (s,t)-cactus representations which have been constructed during the entire execution of CONSTRUCT. Hence, case (b) can occur at most c^* times. We show that $c^* \leq 4\gamma$.
Note that some of the new cycles may be removed by 2-cycle contractions during the conversion of $(\mathcal{R}, \phi)$ to a CNCR in line 16. However, all the new cycles with length at least three remain in the final cactus representation. Assume that we need to apply a 3-cycle insertion after combining $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ and $(\mathcal{R}_{G_i}, \phi_{G_i})$.
Let z_i be the joint node in the union operation $(\mathcal{R}_{(s,t)}, \phi_{(s,t)}) \oplus (\mathcal{R}_{G_i}, \phi_{G_i})$, and suppose z_i is a 3-junction empty node in the resulting cactus representation. By Lemma 3.7, at most three 2-cycles are contracted after the 3-cycle (y_1, y_2, y_3) is inserted at z_i. Thus, at most three proper cycles in $(\mathcal{R}_{G_i}, \phi_{G_i})$ and the new cycles in $(\mathcal{R}_{(s,t)}, \phi_{(s,t)})$ are eliminated. However, the cycle (y_1, y_2, y_3) is a proper cycle in the resulting cactus representation, and this will never be removed in the final cactus representation for G.

Let γ_3 be the number of 3-cycles in the final cactus representation. The number of proper cycles which are removed by 2-cycle contractions is at most $3\gamma_3$. Therefore, the number of proper cycles c^* that appear during the execution of CONSTRUCT is at most $3\gamma_3 + \gamma \leq 4\gamma$.

The next lemma gives the time-complexity of computing all MA-orderings during the execution of CONSTRUCT.

Lemma 4.6 *The total time to compute all MA-orderings in* CACTUS *during the execution of* CONSTRUCT *is* $O(nm + n^2 \log n)$.

Proof Let p_i denote the total number of times line i of CACTUS is executed for an input graph G. Since an MA-ordering can be found in $O(m + n \log n)$ time, it is sufficient to show that $p_2 = O(n)$.
The input graph G is recursively decomposed into subgraphs by procedure CACTUS until all the resulting graphs are isolated vertices. In other words, the leaves of the decomposition tree correspond to graphs consisting of single vertices. Let q be the number of leaves of the decomposition tree. Clearly, $p_2 = p_4 + p_9 + p_{10}$. In each of lines 4 and 9, two vertices are contracted into one vertex, thereby reducing the number of vertices by one. After executing line 12, the number of vertices in the current graph increases by one. Hence, $q = n - p_4 - p_9 + p_{12}$. If line 12 is executed k times in the same execution of **for** loop of lines 11-14, the number of components increases by $k - 1$. Thus $q = 1 + p_{12} - p_{10}$. Therefore, $n - p_4 - p_9 + p_{12} = 1 + p_{12} - p_{10} \Rightarrow p_4 + p_9 = n - 1 + p_{10}$. Hence, $p_2 = p_4 + p_9 + p_{10} = n - 1 + 2p_{10}$. From Lemma 4.5 and Corollary 3.1, we know that $p_{10} \leq 4\gamma = O(n)$. Hence, the result follows.

Next, we consider the time complexity of computing all maximum flows.

Lemma 4.7 *The total time to compute all maximum flows in* CACTUS *during the execution of* CONSTRUCT *is* $O(\gamma m \log n)$.

Proof By Lemma 4.5, case (b) occurs $O(\gamma)$ times. Thus, the total time to compute the maximum flows $F_{(s,t)}$ in line 6, each of which produces at least one new minimum cut in the (s,t)-cactus representation in line 7, is $O(\gamma m \log n)$. Let's consider the case (a). We

will show that the time complexity for computing the maximum flows $F_{(s,t)}$ in line 6, none of which produces a new minimum cut in the (s,t)-cactus representation in line 7, is also $O(\gamma m \log n)$. Assume that a maximum flow $F_{(s',t')}$is computed in the current graph G'. The time bound to compute the maximum flow is $O(|E(G')| \log |V(G')|)$. We choose an old minimum cut $(V(G') - w, w)$ which is represented by its (s',t')-MC-partition in G'. This minimum cut is eliminated after contracting s' and t', and it will never appear in any graph in the decomposition tree. For the old minimum cut $(V(G') - w, w)$, there is a graph G'', such that its (s,t)-cactus representation represents the minimum cut as a new minimum cut. Thus, there is a subset V_i in the (s,t)-MC-partition $(V_1, ..., V_r)$ of $V(G'')$ such that $V(G'') - V_i$ is contracted into the vertex w to obtain a graph G''_i. Clearly, $|E(G')| \leq 2|E(G''_i)|$ and $|V(G')| \leq 2|V(G''_i)|$. Hence, $O(|E(G')| \log |V(G')|) = O(|E(G''_i)| \log |V(G''_i)|)$. Also, $\sum_{i=1}^{r} |E(G''_i)| \leq 2|E(G'')|$. Since no minimum cut appears in more than two graphs in the decomposition tree, there is at most one other old minimum cut corresponding to the graph G'_i (this is possible only for $r = 2$). Therefore, the time bound to compute the maximum flows which have old minimum cuts corresponding to some graph G''_i of the (s,t)-MC-partition $(V_1, ..., V_r)$ of $V(G'')$ is

$$\sum_{i=1}^{r} O(|E(G''_i)| \log |V(G''_i)|) = O(|E(G'')| \log |V(G'')|).$$

So, the time for computing all maximum flows in case (a) is bounded by the time for computing all maximum flows in case (b), which is $O(\gamma m \log n)$. Hence the lemma follows.

From the above discussion we arrive at the following result.

Theorem 4.1 *Algorithm* CONSTRUCT *constructs a CNCR for all minimum cuts $\mathcal{C}(G)$ of a graph G in $O(nm + n^2 \log n + \gamma m \log n)$ time, where γ is the number of cycles in the resulting CNCR.*

Chapter 5

Conclusion and Future Work

In this project, we have studied the combinatorial structure of minimum cuts of a graph and its cactus representation, and have given a new algorithm to construct the cactus representation. One interesting research area to carry out the work further is to consider the properties of approximate minimum cuts of a graph. An **approximate minimum cut** of a graph is a cut whose size is at most $c\lambda$, where c is the *approximation factor* and λ is the size of a minimum cut of the graph. Alternatively, one can try an altogether different approach like matroids to study the minimum cuts of a graph.

Some of the interesting problems which can be considered next are:

- Generalizing and extending some of the combinatorial properties of minimum cuts to approximate minimum cuts, like submodularity of cuts, crossing cuts lemma and circular partition lemma.
- Building a data structure similar to the cactus, to represent all the approximate minimum cuts of a graph.
- Implementation of some of the algorithms given in this report and in the literature.
- Edmond's theorems on packing arborescences.
- Matroid approach to find the edge-connectivity and all minimum cuts of a graph.
- The Steiner minimum cut problem and its cactus representation.
- Obtaining an efficient algorithm for the Steiner minimum cut problem.

Bibliography

[B75] R. E. Bixby, The minimum number of edges and vertices in a graph with edge connectivity n and m n-bonds, *Networks* **5** (1975), 253-298.

[DKL76] E. A. Dinits, A. V. Karzanov and M. V. Lomonosov, On the structure of a family of minimal weighted cuts in a graph, *Studies in Discrete Optimization* (in Russian), 290-306, 1976.

[F99] L. Fleischer, Building chain and cactus representation of all minimum cuts from Hao-Orlin in the same asymptotic run time, *Journal of Algorithms* **33** (1999), 51-72.

[G93] H. N. Gabow, A representation for crossing set families with applications to submodular flow problems, *Proceedings of the 4th ACM Symposium on discrete algorithms*, 1993, 202-211.

[G95] H. N. Gabow, A matroid approach to finding edge connectivity and packing arborescences, *Journal of Computer and System Sciences* **50** (1995), 259-273.

[GT88] A. V. Goldberg and R. E. Tarjan, A new approach to the maximum flow problem, *Journal of the ACM* **35** (1988), 921-940.

[HO94] J. Hao and J. B. Orlin, A faster algorithm for finding the minimum cut in a directed graph, *Journal of Algorithms* **17** (1994), 424-446.

[KT86] A. V. Karzanov and E. A. Timofeev, Efficient algorithms for finding all minimum edge cuts of a nonoriented graph, *Cybernetics* **22** (1986), 156-162.

[L76] E. L. Lawler, "Combinatorial Optimization: Networks and Matroids", Holt, Rinehart & Winston, New York, 1976.

[NI92] H. Nagamochi and T. Ibaraki, Computing edge connectivity in multigraphs and capacitated graphs, *SIAM Journal of Discrete Mathematics* **5** (1992), 54-66.

[NK94] H. Nagamochi and T. Kameda, Canonical cactus representation for minimum cuts, *Japan Journal of Industrial and Applied Mathematics* **11** (1994), 343-361.

[NK96] H. Nagamochi and T. Kameda, Constructing cactus representation for all minimum cuts in an undirected network, *Operations Research Society of Japan* **39** (1996), 135-158.

[NV91] D. Naor and V. V. Vazirani, Representing and enumerating edge connectivity cuts in RNC, *Proceedings of the Second Workshop on Algorithms and Data Structures*, 1991, LNCS **519** in Lecture Notes in Computer Science, 273-285, Springer Verlag.

[PQ80] J. C. Picard and M. Queyranne, On the structure of all minimum cuts in a network and applications, *Mathematical Programming Study* **13** (1980), 8-16.