# Counting Cutsets Faster

by

Kim Hung Cheung

B.Sc. Simon Fraser University, Burnaby, Canada, 1987



A thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

in the Department of Computer Science

We accept this thesis as conforming
to the required standard

[redacted]

Dr. Wendy Myrvold, Supervisor (Department of Computer Science)

[redacted]

Dr. John Ellis, Department Member (Department of Computer Science)

[redacted]

Dr. Frank Ruskey, Department Member (Department of Computer Science)

[redacted]

Dr. Donald Miller, Outside Member (Department of Mathematics and Statistics)

[redacted]

Dr. Luis Goddyn, External Examiner (Department of Mathematics and Statistics)

# Abstract

A subset $C$ of the edges of a graph $G$ is a *cutset* if $G$ is disconnected when the edges in $C$ are removed. In this thesis, we give two algorithms for counting cutsets faster. Both compute the number of cutsets of size $(\lambda+x)$ in polynomial time for fixed $x$ where $\lambda$ is the minimum cutsize. One algorithm uses inclusion and exclusion of edges and is similar to an algorithm of Colbourn and Ramanathan. However, our algorithm uses $s,t$-edge connectivity in most of the recursive calls rather than edge connectivity as in Colbourn and Ramanathan, hence, it is faster. The second algorithm generalizes a method of Ball and Provan and offers a faster run-time than the one by Colbourn and Ramanathan by a factor of at least $O(n)$.

Examiners:

[redacted]
Dr Wendy Myrvold

[redacted]
Dr John Ellis

[redacted]
Dr Luis Goddyn

[redacted]
Dr Donald Miller

[redacted]
Dr Frank Ruskey

# Acknowledgements

Special thanks are due to my thesis supervisor Wendy Myrvold. She has given me numerous comments and suggestions on writing the thesis. Without her I could not possibly have finished the thesis. Thanks to committee members John Ellis, Luis Goddyn, Donald Miller and Frank Ruskey who spent time and effort reading the thesis.

I dedicate the thesis to my parents for everything.

# Contents

# List of Figures

# Chapter 1

# Introduction

### The Purpose

In this thesis, we present two new algorithms that are designed for counting edge cutsets of any size for a graph. These algorithms are inspired by the algorithms designed by Ball and Provan [2] and Colbourn and Ramanathan [5]. A theoretical analysis shows that these new algorithms are faster than the fastest known algorithm by Colbourn and Ramanathan [5].

### Overview

A subset $C$ of the edges of a graph $G$ is a *cutset* if $G$ is disconnected when the edges in $C$ are removed. The fastest known algorithm for counting the number of minimum cutsets of a graph is that of Ball and Provan [2]. It is difficult to extend this algorithm to count cutsets of non-minimum size. An algorithm that works for all cutset sizes is that of Colbourn and Ramanathan [5], however, it is much slower than the one by Ball and Provan when counting minimum cutsets. We would like an algorithm that has time complexity close to the one by Ball and Provan and as general as the one by Colbourn and Ramanathan.

We present two algorithms to remedy the drawbacks stated. Both algorithms

are as general as the one by Colbourn and Ramanathan. NUM_CUTS, one of the new algorithms, runs faster than Colbourn and Ramanathan's algorithm. ECA, the other one, employs the main idea from the Ball and Provan algorithm. Its worst case time complexity is at least $\Omega(n)$ faster than the Colbourn and Ramanathan algorithm. However, it is slower than Ball and Provan's minimum cutset counting algorithm.

The algorithms considered in this thesis have exponential time complexity when used to compute cutsets of arbitrary size. This is not surprising since counting cutsets of arbitrary size is $\sharp P$-complete [5].

The second chapter explains the background knowledge and notations that are necessary to understand this thesis. The third chapter presents a simple recursive algorithm for counting cutsets. This algorithm is crucial to the understanding of the algorithms in chapter four, five and seven. The fourth chapter describes Colbourn and Ramanathan's counting cutsets algorithm. The fifth chapter describes a new algorithm, NUM_CUTS, that counts cutsets faster than Colbourn and Ramanathan's algorithm. The sixth chapter describes Ball and Provan's minimum cutset counting algorithm. The seventh chapter describes a new algorithm, ECA, that is at least a factor of $\Omega(n)$ faster than Colbourn and Ramanathan's when counting cutsets of any size. The eighth chapter summarizes the results. The ninth chapter suggests some future enhancements.

# Chapter 2

# Background

## 2.1 Terminology for Graph

### 2.1.1 Definition of a Graph

An *undirected graph* is an ordered triple denoted as $G = (V(G), E(G), \psi_G)$ consisting of a nonempty set $V(G)$ of vertices, a set $E(G)$, disjoint from $V(G)$, of edges, and an *incidence function* $\psi_G$ that associates with each edge of $G$ an unordered pair of vertices of $G$. The number of vertices $|V(G)|$ is $n$. The number of edges $|E(G)|$ is $m$. If $e$ is an edge and $u$ and $v$ are vertices such that $\psi_G(e) = (u, v)$, then $e$ is said to be *incident to* $u$ and $v$. A *directed graph* is an ordered triple denoted as $G' = (V(G'), E(G'), \psi_{G'})$ consisting of a nonempty set $V(G')$ of vertices, a set $E(G')$, disjoint from $V(G')$, of arcs, and an incidence function $\psi_{G'}$ that associates with each arc of G' an ordered pair of vertices of G'. If $e$ is in $E(G')$ and $u$ and $v$ are vertices such that $\psi_{G'}(e) = (u, v)$, $e$ is said to *join* $u$ to $v$. A *loop* $e$ of $G$ is an edge such that $\exists u \in V(G)$, $\psi_G(e) = (u, u)$. *Multiple edges* $e_1, e_2, \ldots e_i$ are edges such that $\exists u, v \in V(G)$, $\psi_G(e_j) = (u, v)$, for $1 \leq j \leq i$. A graph is *simple* if it does not have loops and multiple edges. The incidence function of a simple

graph is an one-to-one mapping In this case, we simplify the notation by stating $e = (u, v)$ instead of $\psi_G(e) = (u, v)$ A graph is *trivial* if it contains only one vertex A *path* from vertex $v_1$ to $v_i$ is an alternating sequence of vertices and edges, $v_1\, e_1\, v_2 \ldots e_{i-1}\, v_i$, where $e_j$ is an edge between $v_j$ and $v_{j+1}$ A vertex $u$ is *connected to* a vertex $v$ if there is a path from $u$ to $v$ A graph is *connected* if every pair of vertices is connected, otherwise, it is *disconnected* A *spanning subgraph* of $G$ is a graph $H = (V(G), E(H), \psi_H)$ where $E(H) \subseteq E(G)$ and $\psi_H(e) = \psi_G(e)$, if $\forall e \in E(H)$ A *cycle* is a path $v_1\, e_1\, v_2 \ldots e_{i-1}\, v_i\, e_i\, v_1$, where $v_j \neq v_k$ for $1 \leq j, k \leq i$ An *acyclic* graph is one that contains no cycles A *tree* is a connected acyclic graph A *spanning tree* of a graph $G$ is a spanning subgraph of G that is a tree

### 2.1.2 Cutsets

A *cutset* $C$ of a connected graph $G$ is any subset of the edges of $G$ whose removal leaves $G$ disconnected A *minimum cutset* is a cutset of smallest size The size of a minimum cutset, called the *edge-connectivity* of $G$, is denoted by $\lambda$ Given a cutset of size $\lambda + x$, $x \geq 0$, the value of $x$ is called the *excess* of the cutset A *minimal cutset* is a cutset such that a graph cannot be disconnected by removing any proper subset of edges in the cutset For instance, a minimum cutset is a minimal cutset A minimal cutset of a non-trivial graph can be defined as the set of edges $(u, v)$ where $u \in S$ and $v \in \overline{S}$ for some sets $S \subset V(G)$ and $\overline{S} = V(G) - S$ We denote such a cutset by $(S,\overline{S})$ In this thesis, we need to examine more closely the graph obtained by removing some subset $E'$ of the edges from a graph $G$ where $E'$ is commonly a cutset of $G$ This graph $H = (V(G), E(G) - E', \psi_H)$ is denoted simply by $G - E'$, where $\psi_H(e) = \psi_G(e)$, if $\forall e \in E(G) - E'$ When a graph is disconnected, any subset of the edges including the empty set is a cutset

An *$s,t$-cutset* is a cutset such that the vertices $s$ and $t$ are disconnected by removing edges in the $s,t$-cutset from $G$ The size of a minimum $s,t$-cutset, $\lambda_{st}$, is

called the *s,t-edge connectivity*

The number of cutsets of size $i$ is denoted by $N_i$. Since there is a one-to-one correspondence between $i$-edge cutsets and $(m-i)$-edge disconnected spanning subgraphs of a graph, $N_i$ is also the number of disconnected subgraphs of size $m-i$.

### 2.1.3 Some Graph Operations

**Edge Deletion $G-e$**

Given a graph $G$ and an edge $e$ in $G$, we define $G-e$ as $(V(G), E(G)-\{e\}, \psi_{G-e})$, where $\psi_{G-e}(e') = \psi_G(e')$, if $\forall e' \in E(G)-\{e\}$. We call this operation *edge deletion*. We define edge deletion to be left associative. For example, $G-e_1-e_2$ is $(G-e_1)-e_2$.

**Edge Contraction $G \cdot e$**

Given a graph $G$ and an edge $e=(u,v)$ in $G$, we define $G \cdot e$ as $(V(G)-\{v\}, E(G)-\{e\}, \psi_{G \cdot e})$ where $\psi_{G \cdot e}$ is defined as follows.

$$\begin{aligned}
\psi_{G \cdot e}(e') &= \psi_G(e') \quad \text{if } \forall e' \in E(G)-\{e\} \text{ and } \forall w \in V(G),\ \psi_G(e') \neq (w,v) \\
&= (w,u) \quad \text{if } \forall e' \in E(G)-\{e\} \text{ and } \forall w \neq v \in V(G),\ \psi_G(e') = (w,v) \\
&= (u,u) \quad \text{if } \forall e' \in E(G)-\{e\},\ \psi_G(e') = (v,v)
\end{aligned}$$

The above definition of $G \cdot e$ contracts vertices $u$ and $v$ to a new vertex which is labeled $u$. Examples of contracting an edge in a simple graph and a graph with multiple edges are given in Figure 2.1. We call this operation *edge contraction*. We define edge contraction to be left associative. For example, $G \cdot e_1 \cdot e_2$ is $(G \cdot e_1) \cdot e_2$.

**Lemma 2.1.1** *The edge set $C$ is a cutset of $G$-$e$* if and only if *$C \cup e$ is a cutset of $G$.*

Proof. If $C$ is a cutset of $G-e$, then $(G-e)-C$ is disconnected. Since $G-(C \cup e) = (G-e)-C$, $G-(C \cup e)$ is disconnected and hence $(C \cup e)$ is a cutset of $G$.

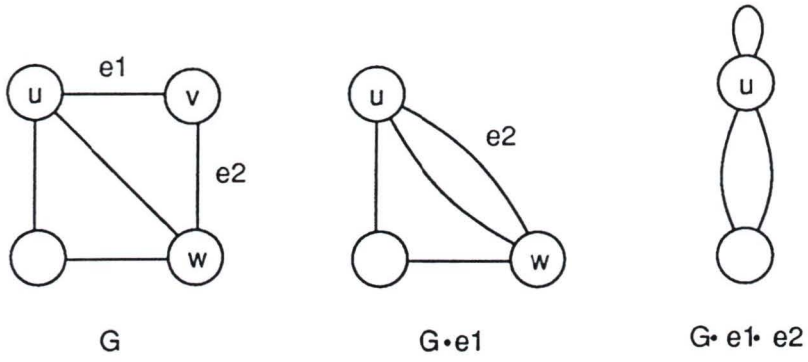



Figure 2.1 Edge Contraction

If $C \cup e$ is a cutset of $G$, it is obvious that $(G - e) - C$ is also disconnected and hence $C$ is a cutset of $G - e$. □

Similarly, we have the following theorem for $G \cdot e$.

**Lemma 2.1.2** *The edge set $C$ is a cutset of $G \cdot e$ if and only if $C$ is a cutset of $G$ and $e$ is not in $C$.*

Proof: If $C$ is a cutset of $G \cdot e$, $e \notin C$ and $(G \cdot e) - C$ is disconnected. Let $(S,\overline{S}) \subseteq C$ be a minimal cutset of $G \cdot e$. Suppose $G - C$ is connected. There must exist an edge $e_2 \in E(G - C)$ such that $e_2$ is incident to a vertex in $S$ of $G$ and another vertex in $\overline{S}$ of $G$. Thus, $e_2 \in (S,\overline{S}) \subseteq C$. That is impossible because $e_2 \notin G - C$. If $C$ is a cutset of $G$ and $e$ is not in $C$, then $G - C$ is disconnected. Since $e$ is not in $C$, it is obvious that $(G \cdot e) - C$ is disconnected. □

### Node Contraction $G \cdot \{u,v\}$

Given a graph $G$ and two vertices $u$ and $v$ in $G$, we define $G \cdot \{u,v\}$ as $(V(G) - \{v\}, E'(G), \psi_{G \cdot \{u,v\}})$, where

$E'(G) = \{e \mid \forall e \in E(G), \psi_G(e) \neq (u,v), \psi_G(e) \neq (u,u)$ and $\psi_G(e) \neq (v,v)\}$, and

$$\begin{aligned} \psi_{G \cdot \{u,v\}}(e) &= \psi_G(e) \quad \text{if } \forall e \in E'(G) \text{ and } \forall w \in V(G), \psi_G(e) \neq (w,v), \\ &= (w,u) \quad \text{if } \forall e \in E'(G) \text{ and } \forall w \neq v \in V(G), \psi_G(e) = (w,v) \end{aligned}$$

The above definition of $G \cdot \{u, v\}$ contracts vertices $u$ and $v$ to a new vertex which is labeled $u$. Examples of contracting $u$ and $v$ in a simple graph are shown in Figure 2.2. We call this operation *node contraction*. We define node contraction be left associative. For example, $G \cdot \{u, v\} \cdot \{u, w\}$ is $(G \cdot \{u, v\}) \cdot \{u, w\}$.

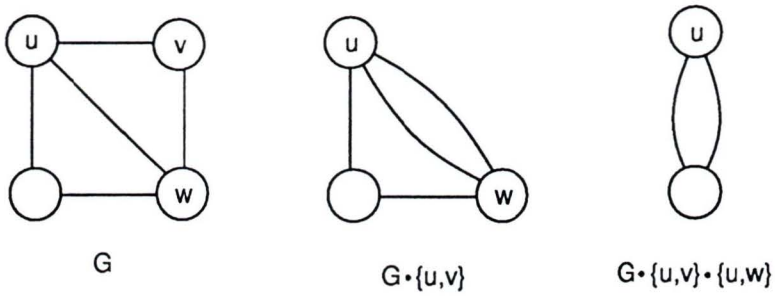



Figure 2.2: Node Contraction

## 2.2 Minimum cut/Maximum flow computation

A minimum cut/maximum flow algorithm is a major component of all the algorithms described in this thesis. In this section, we give some background for the minimum cut/maximum flow problem.

### 2.2.1 $s, t$-flow Computation

In this thesis, a *flow-network* $G_F = (G', s, t, c)$ consists of a directed graph $G'$ with no-self loops, two specific vertices $s$ and $t$ in $V(G')$ called respectively the *source* and the *sink*, and a capacity function $c$ that maps arcs in $G'$ to 0 or 1.

Given an undirected graph $G$, we like to construct a flow-network $G_F$. Since flow-network is defined on a directed graph $G'$ with no-self loops, we first have to define such a graph with respect to $G$.

A directed graph $G'$ in $G_F$ with respect to $G$ is $(V(G), E(G'), \psi_{G'})$, where $E(G') = \{e_{to}, e_{from} \ \forall e \in E(G), \psi(e) \neq (u, u)$ for some $u\}$, and

$$\psi_{G'}(e_{to}) = (u,v) \quad \text{if } e \in E(G) \text{ and } \psi_G(e) = (u,v)$$
$$\psi_{G'}(e_{from}) = (v,u)$$

The flow-network $G_F$ corresponding to $G$ is defined as follows. $G'$ is constructed as defined above, $s$ and $t$ are two vertices in $G$, $c(e_{to}) = c(e_{from}) = 1$, $\forall e_{to}, e_{from} \in E(G')$

The flow-network $G_F$ with respect to $G$ can be used for minimum cut/maximum flow algorithm [6]. It is well-known that a minimum $s,t$-cutset of $G$ can be found after maximizing flow from $s$ to $t$ in $G_F$.

An *$s,t$-flow computation* is defined as follows. A flow function $f$ is an assignment of a non-negative integer $f(e)$ to each arc $e \in G'$, such that the following two conditions hold.

1. For every arc $e \in E(G')$, $0 \leq f(e) \leq c(e)$
2. Let $\alpha(v)$ and $\beta(v)$ be the sets of incoming arcs and outgoing arcs for vertex $v$. The flow from the source $s$ to the sink $t$, satisfies lemma 2.2.1.

**Lemma 2.2.1** *For every $v \in V(G') - \{s,t\}$,*

$$\sum_{e \in \alpha(v)} f(e) = \sum_{e \in \beta(v)} f(e)$$

The maximum *total flow* $F$ of $f$ from $s$ to $t$ is defined by

$$F = \sum_{e \in \alpha(t)} f(e) - \sum_{e \in \beta(t)} f(e)$$

Note that $F$ is the net flow entering $t$.

An *$s,t$-flow computation* is to compute $F$ in $G_F$.

An *auxiliary graph* is used to keep track of potential avenues for pushing further flow through a network. Given a flow-network $G_F$ and flow function $f$, the auxiliary

graph is defined as $(G', c')$ where $c'$ is a capacity function defined as follows For each arc $e = (u, v) \in E(G')$ with $f(e) < c(e)$, the auxiliary graph has an arc $(u, v)$ with capacity $c'(e) = c(e) - f(e)$ and an arc $(v, u)$ with capacity $c'((v, u)) = f(e)$ This implies $c(e) - f(e)$ flow units can be pushed from $u$ to $v$ and $f(e)$ flow units can be pulled from $v$ to $u$ to cancel the flow on arc $(u, v)$

Ford and Fulkerson [9] suggested the use of augmenting paths to change a given flow function in order to increase the total flow An *augmenting path* is a directed path from $s$ to $t$ in an auxiliary graph where each arc on the path has non-zero capacity Such a path can be used to advance flow from $s$ to $t$ Ford and Fulkerson summarize the result of an $s, t$-flow computation in the following maximum flow/minimum cut theorem

**Theorem 2 2 2** *The maximum total $s, t$-flow of a flow-network is equal to the capacity of a minimum $s, t$-cut*

### 2.2.2 Maximum Auxiliary Graph

A *maximum auxiliary graph $U$* is an auxiliary graph derived from a flow-network $G_F$ and its maximum total flow The justification of a maximum auxiliary graph is by the following theorem

**Theorem 2 2 3** *The flow in $G_F$ is maximum if and only if there are no directed $s, t$-paths in the auxiliary graph*

A minimum $s, t$-cutset $(S, \overline{S})$ of $G$ can be derived from $U$ by letting $S$ be the set of vertices reachable from $s$ along a directed path where its arcs have non-zero capacities in the maximum auxiliary graph This set $S$ can be computed by any depth first search algorithm

### 2.2.3 Edge Connectivity

For a fixed vertex $s$ of $G$ where $G$ is non-trivial, any minimal cutset $C$ separates $s$ from a vertex $t$ in $G - C$. Thus, a minimum cutset of $G$ has size

$$\lambda = \min_{t \in V - \{s\}} (s,t\text{-edge connectivity of G})$$

### 2.2.4 Time Complexity of an $s,t$-flow Computation

In the literature, the time complexity of an $s,t$-flow computation for finding a minimum $s,t$-cutset is a function of $n$ and $m$. For example, Dinic's Algorithm with a simple graph and 0/1 capacities is $O(mn^{2/3})$. This time complexity analysis assumes that a maximum auxiliary graph is constructed from scratch. In this thesis, some algorithms re-use auxiliary graph to compute $s,t$-edge connectivity. To establish a new function for comparing time complexity of $s,t$-flow computation, we introduce the following naive algorithm for computing total flow in $G_F$.

Repeat

1. Find an augmenting path in $G_F$.
2. Push one unit of flow along the augmenting path from $s$ to $t$.

Until no augmenting paths in $G_F$.

This naive algorithm is correct because of Theorem 2.2.3. Any depth-first search algorithm can construct an augmenting path in $G_F$ if it exists. The search takes $O(m)$ time. Thus, the time complexity of this algorithm is the product of $O(m)$ and the number of augmenting paths in an auxiliary graph. Any $s,t$-flow computation takes time at least $\Omega(m)$, since all edges must be inspected. To push constant $c$ amount of $s,t$-flow, the naive algorithm requires $O(cm) = O(m)$ time. Thus, if the amount of $s,t$-flow to be pushed is constant, the naive algorithm has $\Theta(m)$ time complexity. In this case, the flow algorithm is not relevant. We prefer to leave the time complexity for non-constant amount of flow as a function so analysis is independent of $s,t$-flow algorithm chosen. In this thesis, we refer to the time

complexity of an $s,t$-flow computation as $A(\lambda_{st} - k)$, where $\lambda_{st} - k$ is the number of augmenting paths in the auxiliary graph that we expected to find and $k$ is the number of augmenting paths already been found. We define $A(0)$ as $O(1)$. Thus, $A(\lambda_{st} - k)$ is $O((\lambda_{st} - k)m)$ assuming naive algorithm. But, we do not want to rule out faster approaches when $k$ is not a constant.

## 2.3 Terminology for Recursive Algorithms

The algorithms which we describe are recursive. In this section, we describe the terminology used when analyzing these algorithms. A call which does not invoke itself recursively is called a *terminal* call, otherwise, the call is an *internal* call.

To describe the relationship among recursive subroutine invocations, we define a tree where there is a node for each subroutine invocation. A node $A$ is defined to be a child of another node $B$, if the call associated with $A$ is invoked by the call which $B$ represents. We refer to this representation of the recursive calls as the *recursive call structure*. This recursive call structure is a tree. A special case of this is a *binary tree*, it is a tree such that each internal node has at most two children. The *height* of a (binary) tree is the number of edges in the longest path from the root to a leaf.

In this thesis, the recursive call structures of some algorithms are binary trees, moreover, in these binary structures, if the call associated with a node $A$ is invoked with the graph $G$, its two children are invoked with $G - e$ and $G \setminus e$ respectively, for some edge $e$. If a call is invoked with $G - e$, we call it a *deletion-call* and if it is invoked with $G \setminus e$ then it is a *contraction-call*.

## 2.4 Network Reliability

### 2.4.1 Definition of a Network

In this thesis, a *network* is a set of computer systems that are linked together to communicate. We assume that computer systems do not fail and each link between two computers operates with an equal independent probability $p$. When all computer systems can communicate with each other, we say this network is *operational*.

### 2.4.2 Modeling of a Network

A *probabilistic graph* is a simple graph where each edge is assigned with an equal independent probability $p$ to determine if it is in the graph. This probabilistic graph is used to model a network. The *nodes* of the probabilistic graph represent computer systems and the *edges* represent the physical links.

With this model, we can derive the probability that the network is operational. This probability is equal to the probability that the probabilistic graph modeling the network is connected. This model of network reliability is called the *all-terminal reliability* of a network. Colbourn's monograph [1] contains a survey about all-terminal reliability.

### 2.4.3 Calculating Network Reliability

#### Network Reliability in terms of $N_i$

The probability that probabilistic graph $G$ is disconnected can be expressed as a sum over all possible disconnected subgraphs of the corresponding probabilistic graph $G$ of the probability that the current state of the network is given by this subgraph. There are $N_i$ disconnected subgraphs of size $m - i$. Each such subgraph occurs with

## 2.4 Network Reliability

### 2.4.1 Definition of a Network

In this thesis, a *network* is a set of computer systems that are linked together to communicate. We assume that computer systems do not fail and each link between two computers operates with an equal independent probability $p$. When all computer systems can communicate with each other, we say this network is *operational*.

### 2.4.2 Modeling of a Network

A *probabilistic graph* is a simple graph where each edge is assigned with an equal independent probability $p$ to determine if it is in the graph. This probabilistic graph is used to model a network. The *nodes* of the probabilistic graph represent computer systems and the *edges* represent the physical links.

With this model, we can derive the probability that the network is operational. This probability is equal to the probability that the probabilistic graph modeling the network is connected. This model of network reliability is called the *all-terminal reliability* of a network. Colbourn's monograph [1] contains a survey about all-terminal reliability.

### 2.4.3 Calculating Network Reliability

#### Network Reliability in terms of $N_i$

The probability that probabilistic graph $G$ is disconnected can be expressed as a sum over all possible disconnected subgraphs of the corresponding probabilistic graph $G$ of the probability that the current state of the network is given by this subgraph. There are $N_i$ disconnected subgraphs of size $m - i$. Each such subgraph occurs with

probability $p^{m-i}(1-p)^i$. Hence, the probability that $G$ is disconnected is

$$\sum_{i=\lambda}^{m} N_i\, p^{m-i}(1-p)^i$$

This formula is called the *unreliability polynomial* of G. Thus, the all-terminal reliability is

$$Rel(G) = 1 - \sum_{i=\lambda}^{m} N_i\, p^{m-i}(1-p)^i$$

The $N_i$ are called the *coefficients* of the unreliability polynomial.

### 2.4.4 Time Complexity of All-terminal Reliability

Ball and Provan [3] have shown that computing the all-terminal reliability is $\sharp P$-complete. However, it is possible to compute some of the coefficients of the unreliability polynomial in polynomial time. For computing $N_{\lambda+x}$ with fixed $x$, the algorithm of Colbourn and Ramanathan [5] is polynomial in $n$ but exponential in $x$. The coefficients $N_i$, $i = m, m-1, \ldots, m-(n-1)$ can also be computed in polynomial time. We explain how to do this in the next paragraph.

Since the edge set of a spanning tree is the smallest one that can connect a graph and a spanning tree has $n-1$ edges, we need at least $n-1$ edges to connect a graph. Thus, the coefficients $N_i$, $i = m, m-1, \ldots, m-n+2$, are equal to $\binom{m}{i}$. The value of $N_{m-(n-1)}$ is $\binom{m}{n-1}$ - $\tau(G)$, where $\tau(G)$ is the number of spanning trees of $G$. The number of spanning trees can be computed in $O(n^3)$ time using the Kirchoff matrix-tree theorem [7] [1].

[1] There is also a technique available for computing determinants faster by Strassen [8].

# Chapter 3

# The Basic Count-Cuts Algorithm

The algorithm BASIC_NUM_CUTS presented in this chapter is based on a simple recurrence relation. This relation is also used in further algorithms. The first objective of this section is to establish the correctness of this recurrence relation. In later chapters, we demonstrate how this recurrence relation can be exploited to obtain faster algorithms.

## 3.1 The Basic Algorithm for Counting Cuts

The algorithm presented in Table 3.1 is called *BASIC_NUM_CUTS*. It is based on the operations $G - e$ and $G \cdot e$ defined in Section 2.1.3 and the following theorem.

**Theorem 3.1.1** *Let Num_Cuts(G,k) be the number of cutsets of size k in G. Then for any edge e of G,*

$$Num\_Cuts(G,k) = Num\_Cuts(G-e,k-1) + Num\_Cuts(G \cdot e,k)$$

Proof. The $k$-edge cutsets can be partitioned into two sets, those that contain a particular edge $e$ and those that do not. By Lemma 2.1.1, $Num\_Cuts(G-e,k-1)$ is equal to the number of $k$-edge cutsets that include $e$. By Lemma 2.1.2,

$Num\_Cuts(G - e, k)$ is equal to the number of $k$-edge cutsets that do not include $e$.

□

```
\*
        BASIC_NUM_CUTS (G,k,n,m) returns the number of cutsets of size
        k
        G is a graph with n nodes and m edges
                                                                    *\
BASIC_NUM_CUTS(G,k,n,m)

  1 [Basis]
    (a) If n = 1, G is trivial, return(0)
    (b) If k = 0 and G is connected, return(0)
    (c) If m < k, return(0)
    (d) If G is disconnected, return( (m choose k) )
  2 Choose any edge e of G
  3 [G - e]
    (a) n_delete = BASIC_NUM_CUTS(G-e, k-1, n, m-1)
  4 [G / e]
    (a) n_contract = BASIC_NUM_CUTS(G/e, k, n-1, m-1)
  5 return(n_delete + n_contract)
```

Table 3.1: BASIC_NUM_CUTS Algorithm

## 3.2 Example for Algorithm BASIC_NUM_CUTS

An example illustrating how the algorithm BASIC_NUM_CUTS counts the number of cutsets of size two is shown in Figure 3.1. The figure shows a computation tree for a graph with four nodes and five edges. The edges are labeled as $a$, $b$, $c$, $d$ and $e$. The example only shows the left subtree of the root and the nodes of the tree are

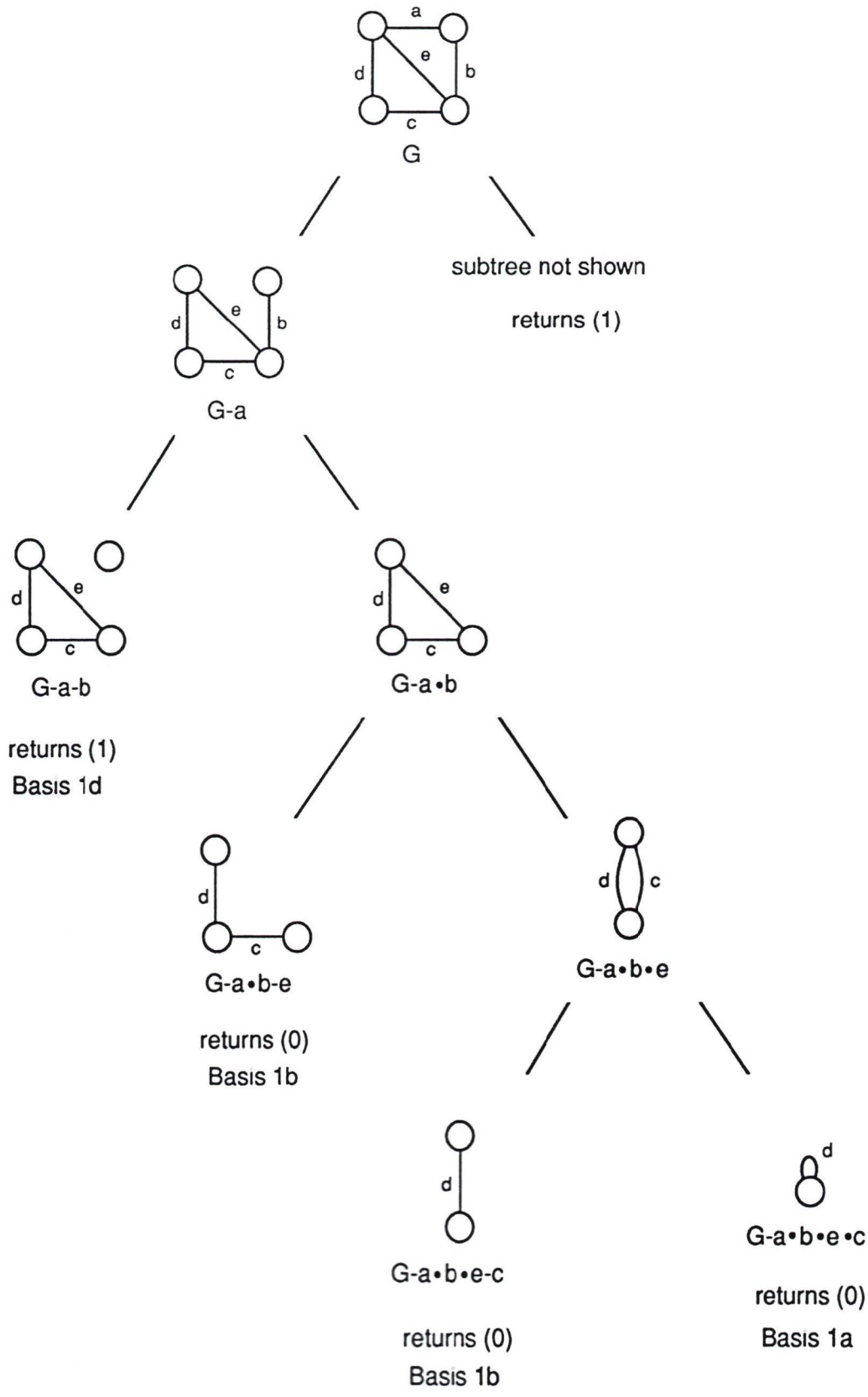



Figure 3.1. An Example for Algorithm BASIC_NUM_CUTS

labeled with the graphs with which the corresponding recursive calls are invoked The algorithm finds one 2-edge cutset $\{a, b\}$ in the left subtree The edges of this cutset are the edges selected for edge deletion along the path from the leaf with $G - a - b$ to the root with $G$

In this example, there are three cases where a call terminates First, a call terminates when a cutset is found, for example the call with $G - a - b$ Second, a call terminates when two edges are deleted from $G$, yet the graph is connected, for example the call with $G - a \;\; b - e$ and the call with $G - a \;\; b \;\; e - c$ Third, a call terminates when the graph is contracted to a single node, for example the call with $G - a \;\; b \;\; e \;\; c$ We should realize that the recursive call structure has internal contraction-calls that return zero, for example the call with $G - a \;\; b$ and the call with $G - a \;\; b \;\; e$ Since the return value from parent is the sum of its children, no cutset is found by the children of these internal contraction-calls

## 3.3 Correctness of the Basic Algorithm

**Theorem 3.3.1** *Algorithm BASIC_NUM_CUTS $(G, k, n, m)$ (Table 3.1) correctly counts cutsets of size $k$ of a graph, where*

*1 $G$ is a graph,*

*2 $n$ is the number of nodes of $G$,*

*3 $m$ is the number of edges of $G$, and*

*4 $k$ is the cutset size required*

Proof $G$ is either connected or not If $G$ is not connected, any subset of edges of size $k$ is a qualified cutset Basis 1(d) computes the number of such subsets

If $G$ is connected, either one of basis 1a, 1b and 1c is executed or no basis condition is satisfied If a basis condition is true, the return value is justified as follows Basis 1a is correct because a trivial graph cannot be disconnected Basis 1b

is correct because $G$ is connected. Basis 1c is correct because the number of edges, $m$, available in $E(G)$ is not enough to construct cutsets of size $k$.

If no basis condition is satisfied, statement 3a and 4a are executed. The initial conditions of a call, BASIC_NUM_CUTS$(G, k, n, m)$, are that $G$ is a graph, $|V(G)| = n$, $|E(G)| = m$, and $k$ is the cutset size required. The call, BASIC_NUM_CUTS$(G - e, k-1, n, m-1)$, at statement 3a satisfies the initial conditions, since $G - e$ is a graph, $|V(G-e)| = n$, $|E(G-e)| = m-1$, and we want to count cutsets of size $k-1$. The call, BASIC_NUM_CUTS$(G \cdot e, k, n-1, m-1)$, at statement 4a satisfies the initial conditions, since $G \cdot e$ is a graph, $|V(G \cdot e)| = n-1$, $|E(G \cdot e)| = m-1$, and we want to count cutsets of size $k$. Statement 3a computes the number of cutsets of size $k-1$ in $G - e$. Statement 4a computes the number of cutsets of size $k$ in $G \cdot e$. According to theorem 3.1.1, the sum of values computed at 3a and 4a is the number of cutsets of size $k$. Therefore, the algorithm counts all the required cutsets.

When $G$ is not connected, Basis 1d is executed, the call terminates. The parameter $m$ of a deletion-call is one less than that of its parent. Since $k \geq 0$, as $m$ decreases, eventually, Basis condition 1c is true. The parameters $n$ and $m$ of a contraction-call are one less than that of its parent. Eventually, $n = 1$, thus, Basis condition 1a is true. The parameter $k$ of a deletion-call is one less than that of its parent. Eventually, Basis condition 1b is true. Eventually, at least one of the basis conditions 1a, 1b, 1c and 1d must apply. Thus, the algorithm terminates eventually. □

## 3.4 Complexity Analysis

The recursive call structure of BASIC_NUM_CUTS is a binary tree. Moreover, the parameter $k$ for a deletion-call is one less than that of its parent and the parameter $n$ for a contraction-call is also one less than that of its parent. Since $n, k \geq 0$ for

all calls, the binary tree representing the recursive call structure has height which is at most $n + k$. Thus, at most $2^{n+k}$ calls are made in total. For each call, a depth first search is required to check if a graph is connected. Hence, the overall time complexity of BASIC_NUM_CUTS is $O(m2^{k+n})$.

# Chapter 4

# Colbourn and Ramanathan's Algorithm

## 4.1 Overview

In the basic algorithm, the branch of the computation tree corresponding to cuts containing a particular edge $e$ is completely explored even when the graph has no cutsets of the required size containing this edge. The Colbourn and Ramanathan algorithm is faster since it avoids this extra work. To accomplish this, the edge for each step is chosen so that it appears in at least one cutset of the required size. This edge selection requires a computation of the edge-connectivity and the construction of a minimum cutset of a graph for each contraction-call.

## 4.2 CR_NUM_CUTS algorithm

Colbourn and Ramanathan's algorithm, CR_NUM_CUTS, is presented in Table 4.1. It has been modified to make comparison with the other algorithms presented in this thesis easier.

*

CR_NUM_CUTS (G,$\lambda$,$x$,n,m) returns the number of cutsets of size $\lambda+x$ in $G$.
$G$ is a graph with $n$ vertices and $m$ edges.
$\lambda$ is the edge-connectivity of G.

*\

CR_NUM_CUTS(G,$\lambda$,$x$,n,m)

1. [*Basis*]
   (a) If $x < 0$, return(0).
   (b) If n = 1, the graph is trivial, return(0).
   (c) If $m < \lambda + x$, return(0).
   (d) If G is disconnected ($\lambda = 0$), return( $\binom{m}{x}$ ).
2. Choose an edge e in a minimum cutset of G.
3. [$G - e$]
   (a) n_delete = CR_NUM_CUTS(G-e,$\lambda$-1,$x$,n,m-1).
4. [$G \cdot e$]
   (a) Find the edge-connectivity $\lambda'$ of G·e.
   (b) n_contract = CR_NUM_CUTS(G·e, $\lambda'$, $x$-($\lambda'$-$\lambda$), n-1, m-1).
5. return(n_delete + n_contract).

Table 4.1 CR_NUM_CUTS Algorithm

### 4.2.1 Example for Algorithm CR_NUM_CUTS

An example illustrating how the algorithm CR_NUM_CUTS counts the number of cutsets of size two is shown in Figure 4 1 The figure shows a computation tree for a graph with four nodes and five edges The labeling of nodes is the same as in Figure 3 1 The algorithm finds the two 2-edge cutsets $\{a, b\}$ and $\{c, d\}$ In this example, a call terminates when either the graph with the call is disconnected or the edge-connectivity of the graph with the call is larger than the cutset size required For example, $G$ $a - c$ $d$ has edge-connectivity two yet we want a cutset of size one and $G$ $a - c - d$ is disconnected

### 4.2.2 Correctness

**Theorem 4 2 1** *CR_NUM_CUTS $(G, \lambda, x, n, m)$ (Table 4 1) finds all the cutsets of a specific size $\lambda + x$, where*

*1 $G$ is a graph,*

*2 $\lambda$ is the edge-connectivity of $G$,*

*3 $n$ is the number of nodes of $G$,*

*4 $m$ is the number of edges of $G$, and*

*5 $x$ is the excess of the minimum cutset size*

Proof The graph $G$ is either connected or not If $G$ is not connected, any subset of edges of size $\lambda + x$ is a qualified cutset Basis 1d computes the number of such subsets

If $G$ is connected, either one of the basis 1a, 1b and 1c is executed or no basis condition is satisfied If a basis condition is true, the return value is justified as follows Basis 1a is correct because when $x < 0$, the size of any cutset of $G$ is large than $\lambda + x$ Basis 1b is correct because a trivial graph cannot be disconnected Basis 1c is correct because the number of edges, $m$, available in $E(G)$ is not enough to

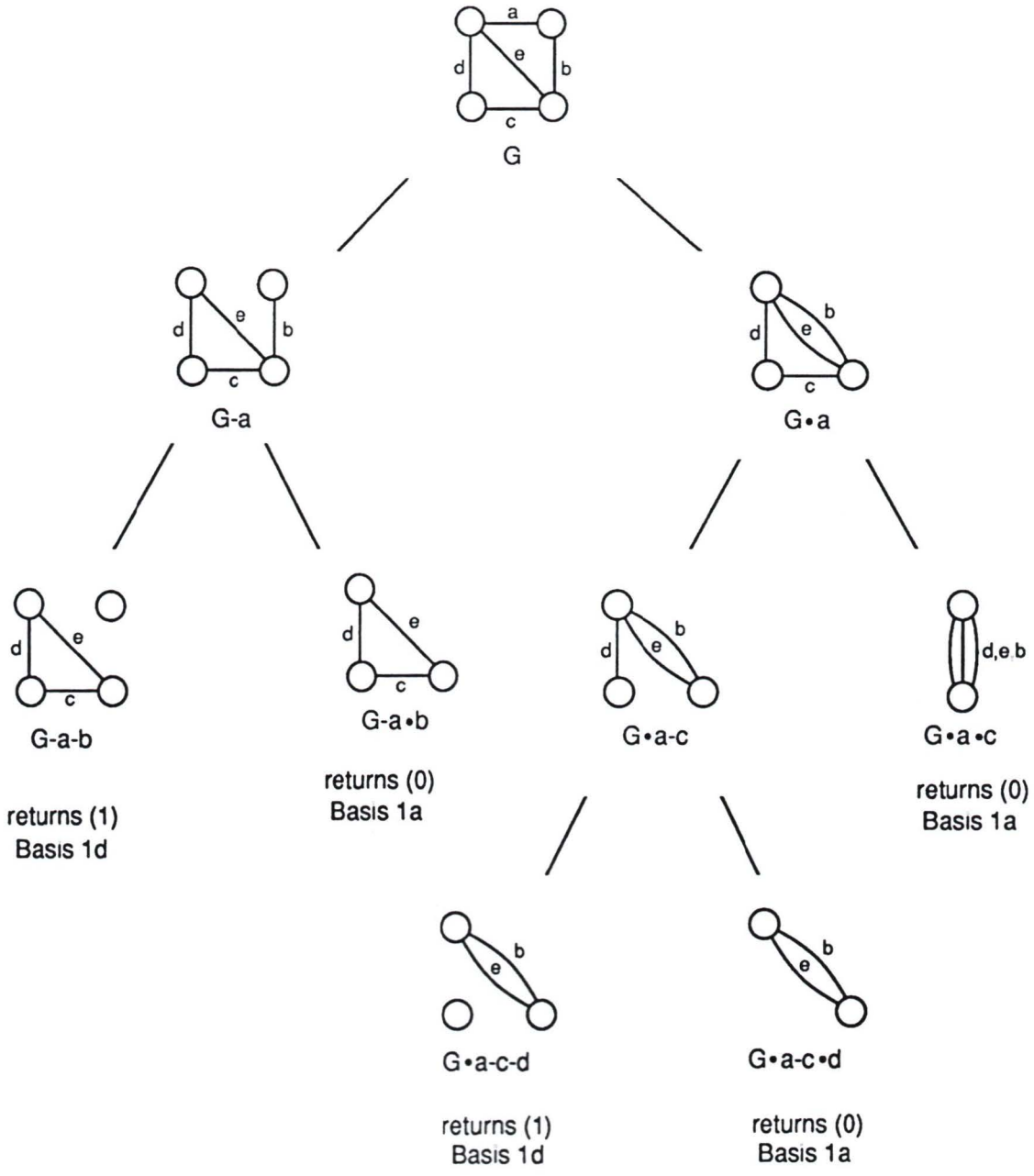



Figure 4.1: An Example for Algorithm CR_NUM_CUTS

construct cutsets of size $k$

If $G$ is connected and no basis condition is satisfied, statement 3a and 4b are executed The initial conditions of a call, CR_NUM_CUTS$(G, \lambda, x, n, m)$, are that $G$ is a graph, $\lambda$ is the edge-connectivity of $G$, $|V(G)| = n$, $|E(G)| = m$, and $x$ is the excess of the minimum cutset size The call, CR_NUM_CUTS$(G - e, \lambda - 1, x, n, m - 1)$, at statement 3a satisfies the initial conditions, because $G - e$ is a graph, $|V(G - e)| = n$, $|E(G - e)| = m - 1$, the edge-connectivity of $G - e$ is $\lambda - 1$, and we want to count cutsets of size $\lambda + x - 1$ in $G - e$ The call, CR_NUM_CUTS$(G \quad e, \lambda', x - (\lambda' - \lambda), n - 1, m - 1)$, at statement 4b satisfies the initial conditions, because $G \quad e$ is a graph, $|V(G \quad e)| = n - 1$, $|E(G \quad e)| = m - 1$, $\lambda'$ is the edge-connectivity of $G \quad e$ computed at statement 4a, and we want to count cutsets of size $(\lambda + x)$ in $G \quad e$ Statement 3a computes cutsets of size $\lambda + x - 1$ in $G - e$ Statement 4b computes cutsets of size $\lambda + x$ in $G \quad e$ According to theorem 3.1 1, the sum of values computed at 3a and 4b is the number of cutsets of size $\lambda + x$ Thus, the algorithm counts all the required cutsets

When $G$ is disconnected, Basis 1d is executed, the call terminates The parameter $m$ of a deletion-call is one less than that of its parent Since $\lambda + x \geq 0$, as $m$ decreases, eventually, Basis condition 1c is true The parameters $n$ and $m$ of a contraction-call are one less than that of its parent Eventually, $n = 1$, thus, Basis condition 1b is true The parameter $x$ of a contraction-call is never larger than that of its parent Basis condition 1a may be true, eventually Eventually, at least one of 1a, 1b, 1c and 1d must apply Thus, the algorithm terminates eventually □

## 4.3 Time Complexity

We first observe that whenever statement 3a is executed, the graph has at least one cutset of the appropriate size (see step 2, Table 4.1) Statement 2 constructs such

a cutset and $e$ is an edge in the cutset. This cutset is counted at statement 3a. Thus, statement 3a returns a non-zero value. Since no deletion-call returns zero, any zero-returning call must be a contraction-call. Since every internal call invokes a deletion-call that returns a non-zero value, every internal call returns a non-zero value. Thus, any contraction-call returning zero must be a terminal call.

We now give an upper bound to the total number of calls made.

**Lemma 4.3.1** *The number of terminal deletion-calls is at most* $N_{\lambda+x}$.

Proof. At least one cutset is enumerated at each of these calls. □

**Lemma 4.3.2** *The number of internal deletion-calls is at most* $(\lambda + x - 1)N_{\lambda+x}$.

Proof. Each deletion-call counts cutset of size one less than that of its parent. Each contraction-call counts cutset of size the same as that of its parent. Hence, the number of internal deletion-calls along the path from a leaf to the root is at most one less the size of a required cutset, $(\lambda + x - 1)$. Hence, by Lemma 4.3.1, there are at most $(\lambda + x - 1)N_{\lambda+x}$ internal deletion-calls. □

**Theorem 4.3.3** *There is at most* $(\lambda + x)N_{\lambda+x}$ *contraction-calls.*

Proof. Each contraction-call has a single sibling which is a deletion-call. There are at most $N_\lambda + (\lambda + x - 1)N_{\lambda+x}$ deletion-calls by Lemmas 4.3.1 and 4.3.2. □

**Theorem 4.3.4** *The Colbourn and Ramanathan algorithm makes at most* $2(\lambda + x)N_{\lambda+x}$ *recursive calls.*

Proof. According to Lemmas 4.3.1, 4.3.2 and 4.3.3, we have $(\lambda+x)N_{\lambda+x}$ contraction-calls and $(\lambda + x)N_{\lambda+x}$ deletion-calls. □

**Theorem 4.3.5** $N_\lambda \leq \begin{pmatrix} n \\ 2 \end{pmatrix}$

Proof. See Lomonosov and Polesskii [10]. □

**Theorem 4 3 6** *$N_{\lambda+x}$ is $O(n^{x+2}m^x)$*

Proof By Colbourn and Ramanathan [5] □

**Corollary 4 3 7** *The number of recursive calls is $O((\lambda + x)n^{x+2}m^x)$*

Proof By Theorem 4 3 4 and theorem 4 3 6 □

If we assume that the edge-connectivity of $G$ is computed by executing at most $(n-1)$ $s,t$−flow computations as indicated in section 2 2 3 and observe that it is not necessary to find an $s,t$-flow of more than $\lambda + x + 1$ units, then the complexity of statement 4a is $(n-1)A(\lambda + x + 1)$, hence, the following theorem follows

**Theorem 4 3 8** *Let $k = \lambda + x$ be the cutset size that we are counting Algorithm CR_NUM_CUTS counts cutsets in time at most*

$$2nA(k+1)\,kN_k = O(A(k+1)\,kn^{x+3}m^x)$$

# Chapter 5

# The Improved Count Cuts Algorithm

The algorithm presented in this chapter is called NUM_CUT. It is faster than CR_NUM_CUTS for two reasons. It does not compute $n-1$ $s,t$-flow computations for every call. As well, it finds the maximum auxiliary graphs for $G-e$ and $G \cdot e$ from the one for $G$ instead of constructing them from scratch.

## 5.1 Improvements on Maximum Auxiliary Graph Construction

A maximum auxiliary graph of a graph is defined in Section 2.2.2. The procedures for finding maximum auxiliary graphs for $G-e$ and $G \cdot e$ from a maximum auxiliary graph for $G$ are stated in Theorems 5.1.1 and 5.1.2.

**Theorem 5.1.1** *Let $(S,\overline{S})$ be a minimum $s,t$-cutset of $G$ obtained from a maximum auxiliary graph for $G$ according to the description in Section 2.2.2, where $S \subset V(G)$ and $\overline{S} = V(G) - S$.*

*If e = (u,v) is in (S,$\overline{S}$), where u ∈ S and v ∈ $\overline{S}$, a maximum auxiliary graph for G-e can be obtained from the one for G by*

*(a) Pushing one unit of flow from u to s*

*(b) Pulling one unit of flow from t to v*

*(c) Removing arcs (u,v) and (v,u)*

Proof Let $G_F$ be the flow-network of $G$ and $U = (G', c')$ be the maximum auxiliary graph for $G$ According to Section 2 2 2, $c'(e) = 0$ because $v$ is not reachable from $u \in S$ using arcs that have non-zero capacities According to section 2 2 1, one unit of flow has been pushed in $U$ from $s$ to $t$ passing through $(u, v)$ and, thus, there is a directed path $P$ in $U$ from $t$ to $s$ passing through the arc $(v, u)$ such that a flow unit can be pulled back Pulling one flow unit along $P$ results in $\lambda - 1$ total flow from $s$ to $t$ in $G_F$, since according to Theorem 2 2 2, maximum total flow in $G_F$ is $\lambda$ Let $U'$ be the resulting auxiliary graph Since no flow has been pushed through $(u, v)$ and $(v, u)$ in $U'$, removing $(u, v)$ and $(v, u)$ from $U'$ does not affect the $\lambda - 1$ total flow in $G_F$ Finally, the flow assignment in $U'$ is feasible for an auxiliary graph for $G - e$ and this flow assignment is maximum It is because $\lambda - 1$ flow units has been pushed from $s$ to $t$ and the corresponding arcs $(u, v)$ and $(v, u)$ of $e$ are removed in $U'$ □

**Theorem 5.1.2** *Let (S,$\overline{S}$) be a minimum s,t-cutset of G obtained from a maximum auxiliary graph for G according to the description in Section 2 2 2, where S ⊂ V(G) and $\overline{S}$ = V(G) − S*
*If e = (u,v) is in (S,$\overline{S}$), where u ∈ S and v ∈ $\overline{S}$, a maximum auxiliary graph for G e can be obtained from the one for G by*

*1 Contracting v into u*

2 *Applying any $s,t$-flow computation routine, until the total flow in the flow-network $G_F$ of $G$ is maximized*

Proof Contracting v into u gives us an auxiliary graph for G e which may not be flow maximized The $s,t$-flow computation routine maximizes the auxiliary graph □

**Run-time improvement**

The next two theorems state the time complexity of updating the auxiliary graph

**Theorem 5 1 3** *Let $(S,\overline{S})$ be a minimum $s,t$-cutset of $G$ obtained from a maximum auxiliary graph for $G$ according to the description in Section 2 2 2, where $S \subset V(G)$ and $\overline{S} = V(G) - S$*
*If $e = (u,v)$ is in $(S,\overline{S})$, where $u \in S$ and $v \in \overline{S}$, a maximum auxiliary graph for $G$-$e$ can be created from the one for $G$ in $O(m)$ time where $m$ is the number of edges of $G$*

Proof According to Theorem 5 1 1, there exists a path $P$ from $s$ to $t$ passing through $e$ A breadth first search can be used to find $P$ □

**Theorem 5.1 4** *Let $(S,\overline{S})$ be a minimum $s,t$-cutset of $G$ obtained from a maximum auxiliary graph for $G$ according to the description in Section 2 2 2, where $S \subset V(G)$ and $\overline{S} = V(G) - S$*
*If $e = (u,v)$ is in $(S,\overline{S})$, where $u \in S$ and $v \in \overline{S}$, a maximum auxiliary graph for G e can be created from the one for $G$ in $O(m\delta)$ time where $m$ is the number of edges of $G$ and $\delta$ is the difference in the $s,t$-edge connectivity of G e against the one for $G$*

Proof Using any standard $s,t$-flow computation algorithm, it takes $O(m)$ time to push one flow unit We must push $\delta$ $s,t$-flow units to get the maximum auxiliary graph for G e Hence, the total time is at most $O(m\delta)$ □

We should note that when $\delta$ is greater than $x$, the excess, we can stop the update of maximum auxiliary graph for G $e$ because the minimum $s,t$-cutset size is too large

## 5.2 The NUM_CUTS Algorithm

In CR_NUM_CUTS, $n$-1 $s,t$-flow computations are done in order to find the edge-connectivity together with an edge in a minimum cutset of a graph This computation has to be carried out for every internal call We are going to show that we can use $s,t$-edge connectivity instead

Suppose we are counting cutsets of size $k$ The main idea of this algorithm is to minimize the number of $s,t$-flow computations in one call and yet we can still bound the number of recursive calls to at most $kN_k$

There are three crucial observations to this algorithm The first observation is that when all minimum $s,t$-cutsets have size greater than $k$, there is no cutset of size $k$ This condition can be checked in $n-1$ $s,t$-flow computations (refer to section 2 2 3) The second observation is that the $s,t$-edge connectivity of $G$ is not larger than that of $G$ $e$ Thus, given a vertex $s$ and a set of vertices $V_1(G)$, which does not include $s$, in $G$ such that for each vertex $t \in V_1(G)$, the $s,t$-edge connectivity of $G$ is larger than $k$ For each vertex $t \in V_1(G)$, the $s,t$-edge connectivity of $G$ $e$ is also larger than $k$ The third observation is that if $e=(s,t)$, we cannot disconnect $s$ and $t$ in $G$ $e$, thus, we have to find a $t' \in V(G) - V_1(G)$, if any, so that the $s,t'$-edge connectivity of $G$ $e$ is not greater than $k$ These observations are crucial to understand this algorithm The algorithm is shown in Table 5 1

*

NUM_CUTS returns the number of cutsets of size $\lambda_{st}+x$ in G G is a graph with n vertices and m edges $\lambda_{st}$ is the minimum $s,t$-cutset size of G for a pair of specified vertices s and t The maximum auxiliary graph of G is stored in st_flow

*\

NUM_CUTS(s,t,st_flow,$\lambda_{st}$,x,n,m)

1 [*Basis*]

(a) If $\lambda_{st} = 0$ and n = 1, the graph is trivial, return(0)

(b) If $m < \lambda_{st} + x$, return(0)

(c) If $\lambda_{st} = 0$, return($\begin{pmatrix} m \\ x \end{pmatrix}$)

2 Choose an edge e of G in a minimum $s,t$-cut (preferably e $\neq (s,t)$ to speed up the algorithm)

3 [$G - e$]

(a) Update st_flow to reflect G-e (as explained in Theorem 5.1.1) and store the result in delete_st_flow

(b) n_delete = NUM_CUTS(s, t, delete_st_flow, $\lambda_{st}$-1, x, n, m-1)

4 [$G \cdot e$]

(a) search_for_t ← FALSE

(b) If e $\neq (s,t)$,
Update st_flow to reflect G·e (as explained in Theorem 5.1.2) and store the result in contract_st_flow
Let $\delta$ be the resulting increase in the $s,t$-edge connectivity

i If $x \geq \delta$,
n_contract = NUM_CUTS(s, t, contract_st_flow, $\lambda_{st} + \delta$, x-$\delta$, n-1, m-1)

ii else search_for_t ← TRUE

(c) If e = $(s,t)$ or search_for_t = TRUE,

i If $\exists$ t′ such that $t'$ has not been considered for $s,t$-flow computation and $\lambda'_{st} \leq \lambda_{st} + x$, n_contract = NUM_CUTS(s, t′, st_flow′, $\lambda'_{st}$, x + $\lambda_{st}$ - $\lambda'_{st}$, n - 1, m - 1), where st_flow′ is a maximum auxiliary graph and $\lambda'_{st}$ is a minimum $s,t'$-cutset for G·e

ii else n_contract = 0

5 return(n_delete + n_contract)

Table 5.1 NUM_CUTS Algorithm

### 5.2.1 Example for Algorithm NUM_CUTS

An example illustrating how the algorithm NUM_CUTS counts the number of cut-sets of size two is shown in Figure 5 1 The figure shows a computation tree for a graph with four nodes and five edges The labeling of nodes is the same as in Figure 3 1 The node $s$ is always selected for $s,t$-flow computation Two shaded nodes in a graph indicate that the $s,t$-edge connectivity is not larger than $k$, for example, the nodes $s$ and $t1$ in $G$ are shaded A node with a cross indicates that the $s,t$-edge connectivity of this crossed node with $s$ is larger than $k$, for example $t2$ in $G$ $a$ is a crossed node and the $s,t2$-edge connectivity is three The graph at the right of an left-arrow is constructed in the call pointed by the arrow This graph is shown to illustrate that no contraction-call is invoked because no $s,t$-cutset has the required size For example, $G$ $a$ $c$ is a graph constructed in the call with $G$ $a$ and the $s,t3$-edge connectivity of $G$ $a$ $c$ is larger than two

The explanation of some recursive call invocations are as follows For the call with $G$, calls are invoked at statement 3(b) and 4(c)i In executing statement 4(c)i, $t2$ of $G$ $a$ is discarded and $t3$ is selected for constructing $s,t$-cutsets of $G$ $a$ For the call with $G$ $a$, only a recursive call is invoked at statement 3(b) Then, in executing statement 4(b), $G$ $a$ $c$ is constructed and the $s,t$-edge connectivity for $G$ $a$ $c$ is found to be greater than $k$ Thus, statement 4(c) is executed As a result of no $t'$ in $G$ $a$ $c$ satisfying $s,t$-cutset requirement, statement 4(c)ii is executed Similarly for $G-a$, $G-a$ $b$ does not have $s,t$-cutsets of the required size, hence, statement 4(c)ii is executed Finally, for the call with $G$ $a-c$, statement 3(b) is executed and a cutset $\{c,d\}$ is constructed Since in executing statement 4(c)i, no $t'$ in $G$ $a-c$ $d$ satisfies the $s,t$-cutset requirement, statement 4(c)ii is executed

The two 2-edge cutsets $\{a,b\}$ and $\{c,d\}$ are constructed in $G-a-b$ and $G$ $a-c-d$

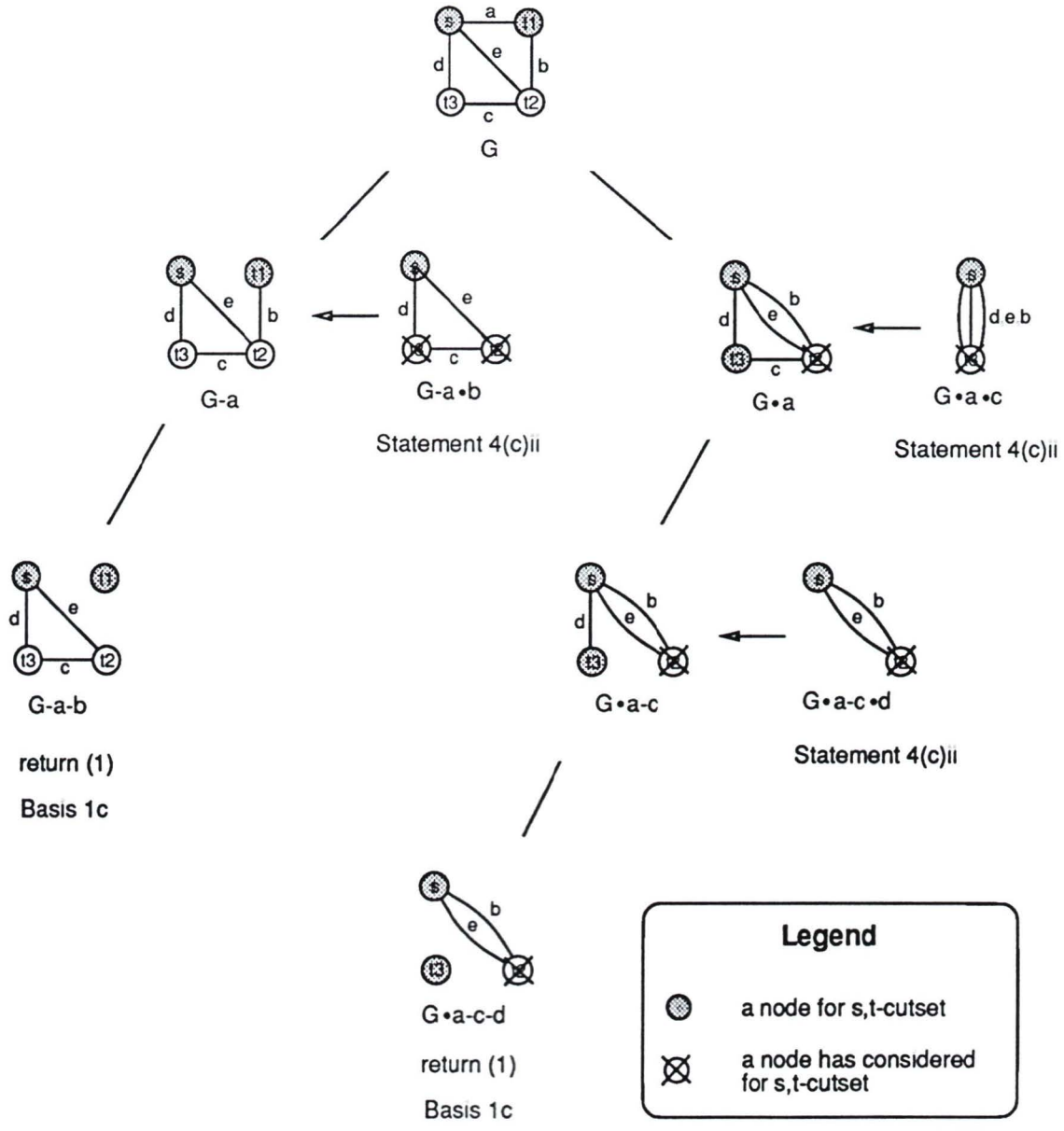



Figure 5.1: An Example for Algorithm NUM_CUTS

## 5.3 Correctness of NUM_CUTS

The computation tree generated from a call to BASIC_NUM_CUTS is a binary tree where each node has zero or two children In NUM_CUTS, each internal call may result in two other calls, or one only There are three cases In the simplest case, the routine is called recursively with G $e$ and G-$e$ statement 3(b) and 4(b)i For the second case, one call for G-$e$ at statement 3(b) and one call at 4(c)i For the third case, one call for G-$e$ at statement 3(b)

**Theorem 5 3 1** *NUM_CUTS(s,t,st_flow,$\lambda_{st}$,x,n,m) (Table 5 1) finds all the cutsets of size $\lambda_{st}+x$, where*

1 *G is a graph with n vertices and m edges*

2 *$\lambda_{st}$ is the minimum $s,t$-cutset size of G for a pair of specified vertices s and t*

3 *The maximum auxiliary graph of G is stored in st_flow*

4 *The number of edges required in addition to $\lambda_{st}$ is $x \geq 0$*

Proof Let $s$ be a fixed node for $s,t$-edge connectivity The graph $G$ is either connected or not If $G$ is not connected, any subset of edges of size $\lambda_{st} + x$ is a qualified cutset Basis 1c computes the number of such subsets

If $G$ is connected, either one of the basis 1a and 1b is executed or no basis condition is satisfied If a basis condition is true, the return value is justified as follows Basis 1a is correct because a trivial graph cannot be disconnected Basis 1b is correct because the number of edges, $m$, available is not enough to construct cutsets of size $\lambda_{st} + x$

If $G$ is connected and no basis condition is satisfied, we have to justify that the recurrence equation in theorem 3 1 1 is implemented correctly to count the number of cutsets of size $\lambda_{st} + x$

The initial conditions of a call, NUM_CUTS$(s,t,st_flow,\lambda_{st},x,n,m)$, are that $s$ and $t$ are vertices in $G$, $st\_flow$ is the maximum auxiliary graph of $G$, $\lambda_{st}$ is the $s,t$-edge connectivity of $G$, $|V(G)| = n$, $|E(G)| = m$, and $x \geq 0$ is the excess of the required cutset. First, statement 3b is executed. The call, NUM_CUTS$(s,t,delete_st_flow,\lambda_{st}-1,x,n,m-1)$, at statement 3b satisfies the initial conditions, since $s$ and $t$ are vertices in $G-e$, $delete\_st\_flow$ is a maximum auxiliary graph for $G-e$ according to statement 3a, the $s,t$-edge connectivity of $G-e$ is $\lambda_{st}-1$, $|V(G-e)| = n$, $|E(G-e)| = m-1$, and we want to count cutsets of size $\lambda_{st}+x-1$ in $G-e$. Thus, cutsets of size $\lambda_{st}+x$ in $G$ that include $e$ are counted at statement 3b.

For those cutsets that do not include $e$, we have to construct them in $G-e$ according to the followings. The edge $e$ is either $(s,t)$ or not. If $e \neq (s,t)$, the $s,t$-edge connectivity of $G-e$ is either greater than $\lambda_{st}+x$ or not. If it is false, there is an $s,t$-cutset of $G-e$ that satisfies our counting requirement. Statement 4(b)i is executed. The call, NUM_CUTS$(s,t,contract_st_flow,\lambda_{st}+\delta,x-\delta,n-1,m-1)$, at statement 4(b)i satisfies the initial conditions, since $s$ and $t$ are vertices in $G-e$, $contract\_st\_flow$ is a maximum auxiliary graph for $G-e$ according to statement 4b, $\lambda_{st}+\delta$ is the $s,t$-edge connectivity of $G-e$ computed at statement 4b, $|V(G-e)| = n-1$, $|E(G-e)| = m-1$, $x-\delta \geq 0$ is true because of statement 4(b)i, and we want to count cutsets of size $(\lambda_{st}+x)$ in $G-e$. In this case, cutsets of size $\lambda_{st}+x$ in $G$ that do not include $e$ are counted at statement 4(b)i.

We now have to consider when $e = (s,t)$ and the condition that the $s,t$-edge connectivity of $G-e$ is greater than $\lambda_{st}+x$. Both of these conditions imply that no $s,t$-cutset, with the specified $s$ and $t$ vertices, in $G-e$ satisfies our cutset size requirement. If there is, indeed, a qualified cutset in $G-e$, there must exists a $t'$ such that the cutset is an $s,t'$-cutset. Statement 4(c)i finds such a $t'$. Since $s,t$-edge connectivity of $G-e$ is never less than that of $G$, statement 4(c)i only checks those

$t'$ that have not been considered for $s,t$-flow computation in those ancestor calls. Either a $t'$ exists such that $s,t'$-edge connectivity is not greater than $\lambda_{st}+x$ or not. If it is true, a call, NUM_CUTS$(s,t',st_flow',\lambda'_{st},x+\lambda_{st}-\lambda'_{st},n-1,m-1)$, is invoked at statement 4(c)i. The call satisfies the initial conditions, since $s$ and $t'$ are vertices in $G-e$, $st\_flow'$ is a maximum auxiliary graph for $G-e$ according to statement 4(c)i, $\lambda'_{st}$ is the $s,t'$-edge connectivity of $G-e$ computed at statement 4(c)i, $|V(G-e)|=n-1$, $|E(G-e)|=m-1$, $x+\lambda_{st}-\lambda'_{st}\geq 0$ is true because of statement 4(c)i, and we want to count cutsets of size $(\lambda_{st}+x)$ in $G-e$. In this case, cutsets of size $\lambda_{st}+x$ in $G$ that do not include $e$ are counted at statement 4(c)i. If no $t'$ that satisfies the condition, the edge-connectivity of $G-e$ is larger than $(\lambda_{st}+x)$. Thus, no call is invoked at statement 4(c)ii.

Since every contraction-call decrements $n$ by one, there is at most $n$ contraction-calls along the path that starts from the root to a leaf in the computation tree. Basis 1a sets a lower bound for $n$. Since every deletion-call decrements $m$ by one, there is at most $m$ deletion-calls along the path that starts from the root to a leaf in the computation tree. The parameter $m$ is decreasing. We have to consider the followings to check $m=0$. If $\lambda_{st}+x>0$, when $m=0$, basis 1b is true. Since $\lambda_{st}$ and $x$ is never negative, when $\lambda_{st}+x\leq 0$, it implies $\lambda_{st}=0$. Thus, if $\lambda_{st}+x\leq 0$, when $m=0$, basis 1c is true. Thus, $m=0$ is checked. At least one of the basis conditions 1a, 1b and 1c is eventually true. Therefore, the algorithm eventually terminates. □

### The Time Complexity of NUM_CUTS

Similar to CR_NUM_CUT, this algorithm only chooses an edge in a cutset that we want to count, we have the following observation. No deletion-call returns zero. Any contraction-call returning zero must be a terminal call. Every internal call returns a non-zero value.

We now give an upper bound to the total number of calls made

**Lemma 5.3.2** *The number of terminal deletion-calls is at most* $N_{\lambda+x}$

Proof At least one cutset is enumerated at each of these calls □

**Lemma 5.3.3** *The number of internal deletion-calls is at most* $(\lambda + x - 1)N_{\lambda+x}$

Proof Each deletion-call counts cutset of size one less than that of its parent Each contraction-call counts cutset of size the same as that of its parent Hence, the number of internal deletion-calls from a leaf to the root is at most one less the size of a required cutset, $(\lambda + x - 1)$ Hence, by Lemma 5.3.2, there are at most $(\lambda + x - 1)N_{\lambda+x}$ internal deletion-calls □

**Theorem 5.3.4** *There is at most* $(\lambda + x)N_{\lambda+x}$ *contraction-calls*

Proof Each contraction-call has a single sibling which is a deletion-call There are at most $N_\lambda + (\lambda + x - 1)N_{\lambda+x}$ deletion-calls by Lemmas 5.3.2 and 5.3.3 □

**Theorem 5.3.5** *The Improved Count Cuts algorithm makes at most* $2(\lambda + x)N_{\lambda+x}$ *recursive calls*

Proof According to Lemmas 5.3.2, 5.3.3 and 5.3.4, we have $(\lambda+x)N_{\lambda+x}$ contraction-calls and $(\lambda + x)N_{\lambda+x}$ deletion-calls □

**Corollary 5.3.6** *The number of recursive calls is* $O((\lambda + x)n^{x+2}m^x)$

Proof By Theorem 5.3.5 and theorem 4.3.6 □

From a leaf to the root, all the calls along this path together perform at most $n - 1$ $s,t$-flow computations at statement 4(c)i Each $s,t$-flow computation takes $A(\lambda_{st} + x + 1)$ time Summing up what have been stated, we have the following theorem

**Theorem 5 3 7** *Let $k = \lambda_{st} + x$ be the cutset size that we are counting The time complexity of NUM_CUTS is at most*

$$2nA(k+1)kN_k = O(A(k+1)kn^{x+3}m^x)$$

Although NUM_CUTS has the same worst case analysis as CR_NUM_CUTS, NUM_CUTS is still faster than CR_NUM_CUTS because it does not compute edge-connectivity for every call

# Chapter 6

# Ball and Provan's Algorithm

## 6.1 How the algorithm works

Ball and Provan's algorithm (BALL_PROVAN) counts minimum cutset It is based on the node contraction operation $G \{s,t\}$ and the following theorem

**Theorem 6 1 1** *Let Num_Min_Cuts(G) be the number of minimum cutsets in G Then for any two nodes s and t,*

$$\text{Num\_Min\_Cuts(G)} = \begin{array}{l}\text{Number of minimum} \\ s,t\text{-cutsets of size } \lambda \text{ (1)}\end{array} + \text{Num\_Min\_Cuts}(G \{s,t\}) \text{ (2)}$$

Proof Any minimum $s,t$-cutset $C$ either disconnects $s$ and $t$ in $G-C$ or not Term 1 counts all minimum $s,t$-cutsets that disconnect $s$ and $t$ in $G-C$ Any minimum cutset $C$ that connects $s$ and $t$ in $G-C$ can be represented by $(S,\overline{S})$ where $s,t \in S$ Since edges that incident to $s$ and $t$ are not in $(S,\overline{S})$, $(S,\overline{S})$ is in $G \{s,t\}$ Obviously, any cutset $C$ in $G \{s,t\}$ is also a cutset in $G$ that connects $s$ and $t$ in $G-C$, if $s$ and $t$ are connected in $G$ □

Suppose we drop the minimum cutset size requirement, can the recurrence equation in theorem 6 1 1 counts cutsets of any size? The proof of theorem 6 1 1 uses

the fact that each cutset to be counted can be represented by $(S,\overline{S})$ However, a non-minimal cutset cannot be represented by $(S,\overline{S})$ We can construct these non-minimal cutsets such that they consist of edges in $E(G) - E(G \ \{s,t\})$, these edges cannot be in $E(G \ \{s,t\})$ Thus, the recurrence equation cannot be directly generalized to compute cutsets of size greater than $\lambda$ An example of such a non-minimal cutset is given in Figure 6 1 The pseudo-code for the Ball and Provan algorithm is given in Table 6 1

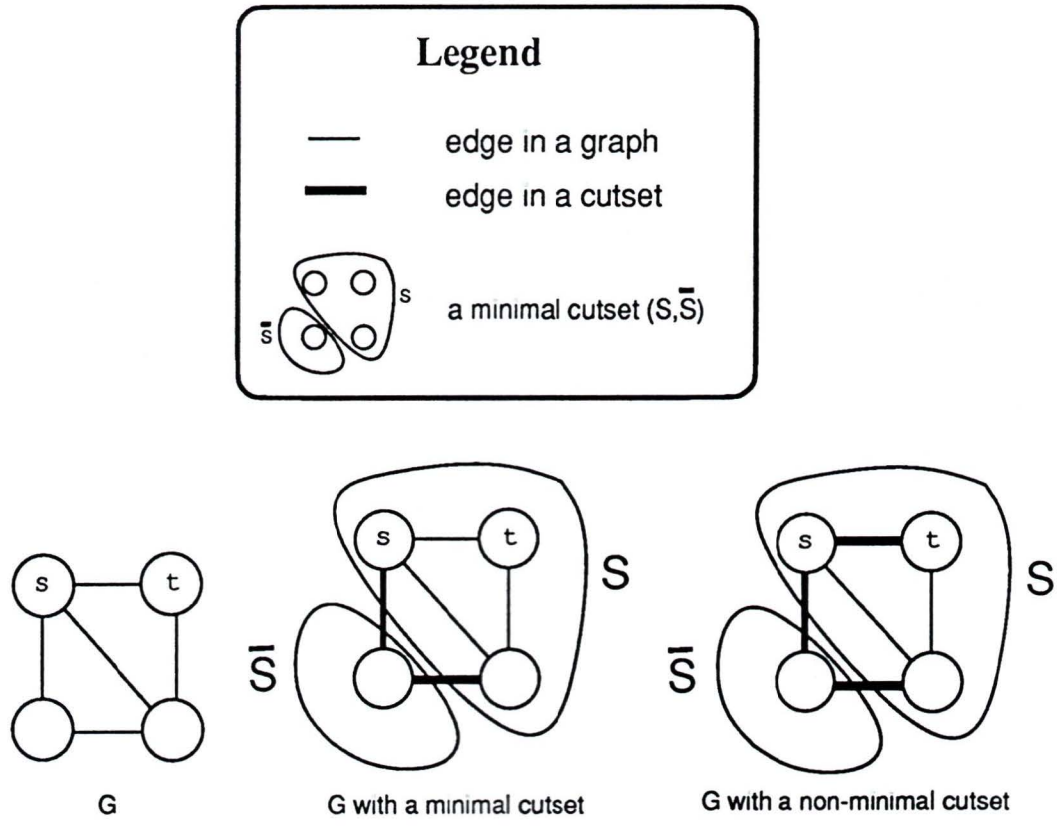



Figure 6 1 An Example of a Non-minimal Cut

### Example for Ball and Provan's Algorithm

An example illustrating how the algorithm BALL_PROVAN computes the number of cutsets of size two is shown in Figure 6 2 Given a graph $G$ that has four nodes

*

BALL_PROVAN returns the number of minimum cutsets of G
G is a graph
s is a fixed node in G
t $\neq$ s is a node in G
$N_\lambda$ is the current sum of minimum cutsets counted
$\lambda_{st}$ is the size of a minimum $s,t$-cutset

*\

BALL_PROVAN(G,s)

1 Find $\lambda$, the size of a minimum cutset in G
Set $N_\lambda = 0$

2 For each t $\in$ G-s do the following

(a) Set $\lambda_{st}$ = size of a minimum $s,t$-cutset

(b) If $\lambda_{st} = \lambda$, set $N_\lambda = N_\lambda$ + NUM_MIN_ST_CUTS(G,s,t),
where NUM_MIN_ST_CUTS returns the number of $s,t$-cutsets of $G$

(c) Set G = G $\{s,t\}$

3 return($N_\lambda$)

Table 6 1 Ball and Provan's Algorithm

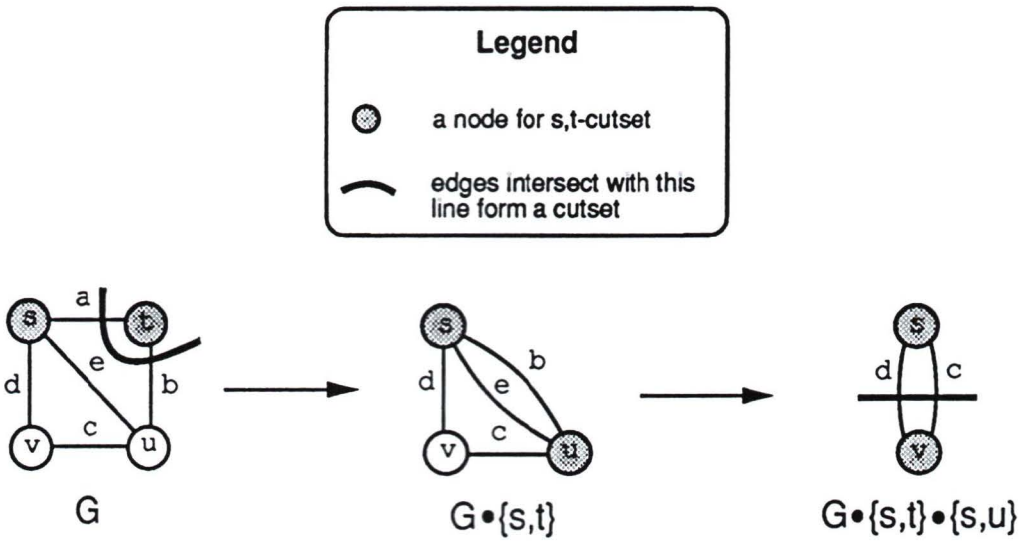



Figure 6 2 An Example for Algorithm BALL_PROVAN

and five edges Edges are labeled as $a$, $b$, $c$, $d$ and $e$ Nodes are labeled as $s$, $t$, $u$ and $v$ A cutset is indicated by those edges intersecting a thick line

First, BALL_PROVAN constructs a minimum $s,t$-cutset $\{a,b\}$ Then, by contracting $t$ to $s$ in $G$, $G$ $\{s,t\}$ is constructed Since the $s,u$-edge connectivity is larger than two No cutset is found Finally, by contracting $u$ to $s$ in $G$ $\{s,t\}$, $G$ $\{s,t\}$ $\{s,u\}$ is constructed A minimum $s,v$-cutset $\{d,c\}$ is constructed

### Run-time Complexity

Ball and Provan's algorithm invokes a subroutine NUM_MIN_ST_CUTS that counts minimum $s,t$-cutsets $n$-1 times The subroutine uses an algorithm that has time complexity $O(mn)$ to compute $s,t$-edge connectivity and derives all $s,t$-cutsets of size $\lambda$ in time $O(mN_\lambda^t)$, where $N_\lambda^t$ is the number of minimum $s,t$-cutsets of size $\lambda$ Thus, NUM_MIN_ST_CUTS computes in $O(mn + mN_\lambda^t)$ Since there are $n$-1 minimum $s,t$-cutset computations and the number of cutsets counted is $N_\lambda$, Ball and Provan's algorithm takes time $O(mn^2 + mN_\lambda)$ Since $N_\lambda$ is $\binom{n}{2}$ [10], the time complexity simplifies to $O(mn^2)$ This approach is the fastest known algorithms for counting minimum cuts

# Chapter 7

# The Extended Contraction Algorithm

## 7.1 The Recurrence Equation for the ECA Algorithm

The extended contraction algorithm (ECA) is a generalization of Ball and Provan's algorithm for counting minimum cutsets. It computes the number of cutsets of any size based on the following recurrence equation.

$$\begin{array}{l}\text{Number of cutsets}\\ \text{of size } k \text{ in G}\end{array} = \begin{array}{l}\text{Number of cutsets C of}\\ \text{size } k \text{ with } s \text{ and } t \text{ in}\\ \text{different components of}\\ \text{G-C (1)}\end{array} + \begin{array}{l}\text{Number of cutsets C of}\\ \text{size } k \text{ with } s \text{ and } t \text{ in the}\\ \text{same component of G-C}\\ \text{(2)},\end{array} \quad (7.1.1)$$

where $s$ and $t$ are any two nodes of the graph $G$.

Suppose the nodes in $G$ are labeled as $t_0, t_1, t_2, \ldots, t_{n-1}$ where $t_0 = s$. The above recurrence equation can be expanded to obtain the following

$$\text{Number of cutsets of size } k \text{ in G} = \sum_{i=1}^{n-1} \begin{array}{l}\text{Number of cutsets } C \text{ of size } k \text{ in G with } s \text{ and} \\ t_i \text{ in different components and } s, t_1, t_2, \ldots, t_{i-1} \\ \text{in the same component of G-C}\end{array} \quad (7.1.2)$$

The correctness of equation 7.1.2 is justified by the following lemmas.

**Lemma 7.1.1** *Equation 7.1.2 enumerates each cutset of size k of G at most once.*

Proof: For $i = i_1$ and $i = i_2$, $i_2 > i_1$, the cutsets enumerated are different, because in step $i_1$, s and $t_{i_1}$ are in different components of $G - C$, and in step $i_2$, s and $t_{i_1}$ are in the same component of $G - C$. □

**Lemma 7.1.2** *Equation 7.1.2 enumerates each cutset of size k of G at least once.*

Proof: For each $k$-edge cutset in $G$, there exists a value $j$ such that $t_1, t_2, \ldots, t_{j-1}$ are on same side of the cutset as $s$, and $t_j$ is not on the same side of the cutset as $s$. Such a cutset is counted when $i$ is equal to $j$. □

## 7.2 Implementing the Recurrence Equation

The recurrence equation 7.1.2 sums $n - 1$ terms. The $i$th term represents the number of cutsets $C$ of size $k$ of $G$ such that in $G - C$, $s$ and $t_i$ are in different components and $s, t_1, t_2, \ldots, t_{i-1}$ are in the same component. Each term is computed by a subroutine COUNT_CONTRACTED_CUTS in ECA. Thus, ECA invokes COUNT_CONTRACTED_CUTS $n - 1$ times to evaluate the recurrence equation 7.1.2. First, we explain COUNT_CONTRACTED_CUTS and then we explain how ECA counts cutsets by invoking COUNT_CONTRACTED_CUTS.

### 7.2.1 Counting Cutsets that Satisfy the Conditions

In this section, we describe COUNT_CONTRACTED_CUTS that counts cutsets C of size $k$ such that in $G - C$, $s$ and $t_i$ are in different components and $s, t_1, t_2, \ldots, t_{i-1}$

are in the same component

The condition that $s, t_1, t_2, \ldots, t_{i-1}$ are in the same component is equivalent to the existence of a Steiner tree for these nodes

**Definition 7.2.1** *A Steiner tree $T$ for a given graph $G$ and a subset $F \subseteq V(G)$ is a subgraph of $G$ such that $F \subseteq V(T)$ and $T$ is a tree. The nodes in $F$ are referred to as Steiner nodes. A Steiner tree $T$ is minimal, if all leaves of $T$ are Steiner nodes.*

To ensure that there always exists a Steiner tree initially, the labeling of the nodes is chosen such that if $F_{i-1} = \{ t_0, t_1, t_2, \ldots, t_{i-1} \}$ then $t_i$ is adjacent to some node in $F_{i-1}$, for all $i = 1,2, \ldots, n-1$. Initially, $F_0 = \{t_0 = s\}$. Such an ordering can easily be found by using a depth-first search algorithm starting at the node $t_0 = s$. This is equivalent to the condition that the subgraph induced by $F_i$ is connected for all $i$.

When the graph $G$ is not connected, COUNT_CONTRACTED_CUTS invokes COUNT_CO_STEINER_SUBSETS to count cutsets $C$ such that the Steiner nodes are connected in $G-C$. First, the correctness of COUNT_CO_STEINER_SUBSETS is presented and then this is followed by that of COUNT_CONTRACTED_CUTS

### Counting Co-Steiner Subsets

If vertices $s$ and $t$ are not connected in $G$, counting $s,t$-cutsets $C$ such that the Steiner nodes are connected in $G - C$ is the same as counting subsets $C_1$ of edges in $G$ such that $G - C_1$ has a Steiner tree for the Steiner nodes. The edge set $C'$ $\subseteq E(G)$ is a *co-Steiner subset* of edges of $G$ with respect to $F$, if there is a Steiner tree for the Steiner nodes in $G - C'$. Thus, when $s$ and $t$ are not connected in $G$, we need to count the number of co-Steiner subsets in $G$ with respect to $F$.

COUNT_CO_STEINER_SUBSETS enumerates co-Steiner subsets using edge inclusion and exclusion. When edge $e = (u, v)$ is contracted in $G$ with the resulting supernode labeled $w$, the set $F$ of Steiner nodes is updated as follows. If at least

*

COUNT_CO_STEINER_SUBSETS returns the number of co-Steiner subsets with respect to $F$ of size $x$ in a disconnected graph $G$
$F$ is a set of Steiner nodes
T is a minimal Steiner tree for $F$
Note When this routine is invoked, there is always a Steiner tree for nodes in $F$

*\

COUNT_CO_STEINER_SUBSETS(G, F, $x$, T)

1 [*Basis*]

(a) If $x = 0$, return(1)

(b) If $| F | = 1$, return( $\begin{pmatrix} | E(G) | \\ x \end{pmatrix}$ )

(c) If $| E(G) |$ - ($| F |$ - 1) $< x$, return(0)

2 [ Counting co-Steiner subsets that do not use edges in E(T) ]
n_subset = $\begin{pmatrix} | E(G) | - | E(T) | \\ x \end{pmatrix}$

3 [ Counting co-Steiner subsets that use at least one edge in E(T) ]
Let the edges in T be $e_1, e_2, \ldots, e_j$
For k = 1 to $j$ do

(a) { use $e_k$ to construct a co Steiner subset }
If G-$e_k$ has a minimal Steiner tree $T'$ for $F$ then
n_subset = n_subset + COUNT_CO_STEINER_SUBSETS(G-$e_k$, F, $x - 1$, $T'$)

(b) { do not use $e_k$ to construct a co-Steiner subset }
G $\leftarrow$ G $e_k$
T $\leftarrow$ T $e_k$
F $\leftarrow$ F$\sim e_k$

4 return(n_subset)

Table 7.1 COUNT_CO_STEINER_SUBSETS Algorithm

one of $u, v$ is a Steiner node in $F$, then $F$ becomes $F - \{u, v\} \cup \{w\}$ Otherwise, $F$ remains unchanged We denote this operation by $F \sim e$ The algorithm is in Table 7.1

### Example of COUNT_CO_STEINER_SUBSETS

An example illustrating how the algorithm COUNT_CO_STEINER_SUBSETS computes the number of co-Steiner subsets of size one is shown in Figure 7.1 The graph $G$ depicted has four nodes and five edges Initially, $T = \{a, b\}$ and $F = \{s, t_1, t_2\}$ Statement 2 computes the number of co-Steiner subsets that do not use edges in $T$ The result is one and the co-Steiner subset is $\{e\}$ Then, statement 3 computes those subsets that use at least one edge in $T$ First, for $G - a$, a new Steiner tree $T' = \{e, b\}$ is constructed An invocation is made with $G - e$, $T'$ and $x - 1$ In this invocation, basis 1(a) returns one co-Steiner subset $\{a\}$ is constructed Returning to the invocation with $G$, $G$ $a - b$ and $T' = \{e\}$ are constructed An invocation is made with $G$ $a - b$, $T'$ and $x - 1$ In this invocation, basis 1(a) returns one co-Steiner subset $\{b\}$ is constructed Finally, $G$ $a$ $b$ and $T' = \emptyset$ are constructed An invocation is made with $G$ $a$ $b$, $T'$ and $x$ In this invocation, basis 1(b) is true no co-Steiner subset is constructed The call terminates

## 7.2.2 Correctness of COUNT_CO_STEINER_SUBSETS

The correctness of COUNT_CO_STEINER_SUBSETS is stated in the following theorem

### Correctness of COUNT_CO_STEINER_SUBSETS

**Theorem 7.2.2** *COUNT_CO_STEINER_SUBSETS($G, F, x, T$) (Table 7.1) finds all co-Steiner subsets with respect to $F$ of size $x$ in a disconnected graph $G$ where $T$ is a minimal Steiner tree for $F$*

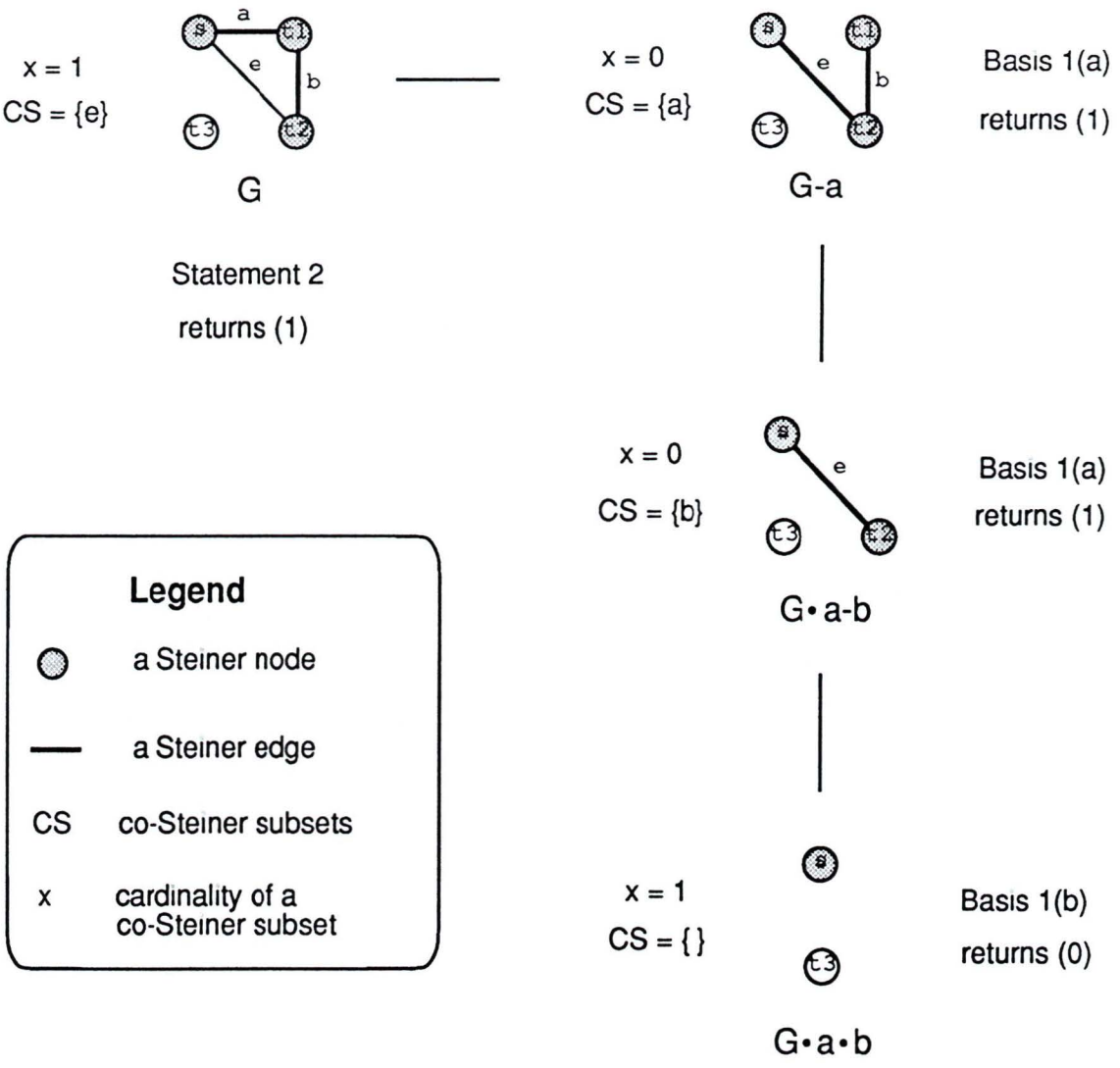



Figure 7.1 An Example for Algorithm COUNT_CO_STEINER_SUBSETS

Proof Either one of the bases 1a, 1b or 1c is executed or no basis condition is satisfied If a basis condition is true, the return value is justified as follows Basis 1a is correct because when $x$ is zero, by definition, an empty set is a cutset Basis 1b is correct because no subset of edges can disconnect a single Steiner node and hence, any subset of edges is a co-Steiner subset Basis 1c is correct because constructing Steiner tree for $F$ requires at least $|F|-1$ edges and there are only $|E(G)|-(|F|-1)$ edges for constructing co-Steiner subsets

When no basis condition is satisfied, the algorithm counts co-Steiner subsets using inclusion and exclusion of the edges in $E(T)$ Either at least one edge in $E(T)$ is used to construct co-Steiner subsets or not If the edges in $E(T)$ are not used in constructing co-Steiner subsets, Steiner nodes in $F$ stay connected by $T$ Thus, any subset of edges of size $x$ selected from $E(G) - E(T)$ is a co-Steiner subset Statement 2 computes the number of such subsets Otherwise, there is a least $k$ for any co-Steiner subset such that $e_1$, $e_2$, , $e_{k-1}$ are not in the subset, but $e_k$ is in the subset These are enumerated at step $k$ of the loop at statement 3

The initial conditions for a call, COUNT_CO_STEINER_SUBSETS$(G, F, x, T)$, are $G$ is a disconnected graph, $F$ is a set of Steiner nodes, $T$ is a minimal Steiner tree for $F$, and $x$ is the size of co-Steiner subsets that we are counting Each call COUNT_CO_STEINER_SUBSETS$(G - e_k, F, x-1, T')$ invoked at statement 3 satisfies the initial conditions This is because $G - e_k$ is a disconnected graph, $F$ is a set of Steiner nodes that are connected by $T'$, a minimal Steiner tree constructed at statement 3a, and we want to count co-Steiner subsets of size $x-1$ The co-Steiner subsets counted by this call do not use edges $e_1$, $e_2$, , $e_{k-1}$ because these edges are contracted in $T$ and $G$ after executing statement 3b for $k-1$ iterations

For termination, $|E(G)|$ and $x$ decrement by one for each recursive call invoked at statement 3a Basis 1a terminates a call when $x = 0$ Basis 1c terminates a call when $|E(G)| - (|F| - 1) < x$ When $k$ is larger than one, $|V(G)|$ decrements

by one for each recursive call invoked at statement 3a Eventually, $|V(G)| = 1$ If $|V(G)| = 1$, $|F| = 1$ Thus, Basis 1b terminates the call when $|V(G)| = 1$ At least one of the basis conditions 1a, 1b and 1c eventually is true Thus, the algorithm terminates □

## COUNT_CONTRACTED_CUTS

We first discuss the theory needed to understand COUNT_CONTRACTED_CUTS This algorithm is based on the basic $G\ e \setminus G - e$ recurrence relation given in Chapter 3 The following lemma is used when choosing an edge $e$ for the recurrence equation to make the algorithm more efficient To state the lemma, we first require the following definition

**Definition 7 2 3** *We assume that the set $F$ has some specified vertex $s$ The vertices in $F$ are contracted together to get the graph $G\ F$ as follows (when we contract these vertices together, the resulting supernode is labeled $s$)*

*The vertex set of $G\ F$ is $V(G) - (F - \{s\})$ The edges of $G\ F$ are of two types*

1. *for each edge $e = (u, v) \in E(G)$ where $u \notin F$ and $v \notin F$, there is a corresponding edge (u,v) in $G\ F$, and*

2. *for each edge $e = (u, v) \in E(G)$ where $u \in F$ and $v \notin F$, there is a corresponding edge (s,v) in $F$*

An example of $G\ F$ is shown in figure 7 2

**Lemma 7 2 4** *Let $G$ be a connected graph Suppose $F \subseteq V(G)$, is a set of nodes such that $s \in F$ and the subgraph induced by $F$ is connected Then for any cutset $C$ such that in G-C*

1. *$s$ and $t$ ($t \notin F$) are in different components, and*

2. *nodes in $F$ are connected by a Steiner tree,*

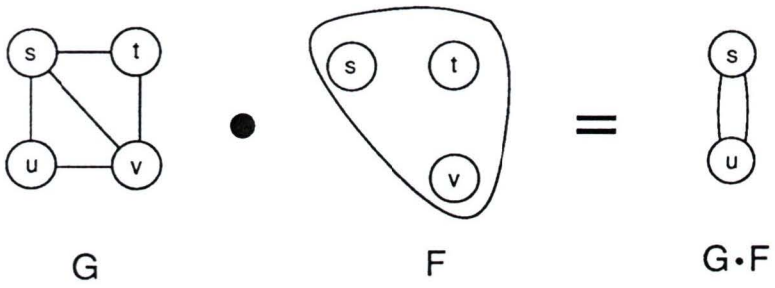



Figure 7.2: An Example of $G \bullet F$

*there exists some $C' \subseteq C$ such that $C'$ is an $s,t$-cutset of $G \bullet F$*

Proof: Let $S$ be the set of vertices in the component of $G - C$ which contains $s$. One such $C'$ for $G \bullet F$ is $(S' = (S - F) \cup \{s\}, \overline{S'})$. □

An example of such a $C'$ in $G \bullet F$ where $C' \subseteq C$ is shown in Figure 7.3.

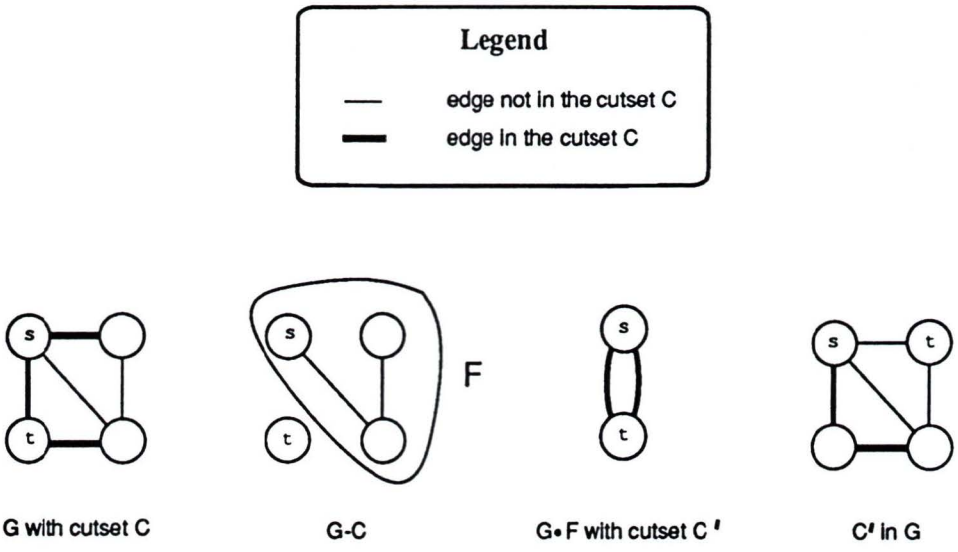



Figure 7.3: An Example of $C'$ in $G \bullet F$

**Corollary 7.2.5** *Let $G$ be a connected graph, $s,t \in V(G)$ and $F \subseteq V(G)$. Nodes in $F$ are connected by a Steiner tree. Let $C'$ be an $s,t$-cutset of $G \bullet F$. Let $C \subseteq E(G)$ be the edges of $G$ which correspond to those in $C'$. Then, $C$ is a cutset of $G$ such that in G-C*

1 *s and t ($t \notin F$) are in different components, and*

2 *nodes in F are connected by a Steiner tree*

Proof Since the edges, $E_1$, induced on $F$ are absent in $G - F$, the edges in $E_1$ are not in $C'$. Since the corresponding $C'$ edges in $G$ is $C$, the corresponding edges in $E_1$ are also not in $C$. Thus, nodes in $F$ are connected in $G - C$ by the corresponding edges in $E_1$. Since $s$ and $t$ are disconnected in $(G - F) - C'$, $s$ and $t$ are disconnected in $G - C$. □

The COUNT_CONTRACTED_CUTS algorithm is in Table 7.2.

### Example of COUNT_CONTRACTED_CUTS

Two examples illustrating how the algorithm COUNT_CONTRACTED_CUTS counts the number of cutsets of size three are shown in Figures 7.4 and Figure 7.5. The graph $G$ has four nodes and five edges. In Figure 7.4, $F = \{s\}$ and $(S,\overline{S}) = \{a,b\}$. COUNT_CONTRACTED_CUTS computes the number of $s,t_1$-cutsets $C$ such that in $G - C$, $s$ and $t_1$ are in different components (note $F$ only has one node $s$). Three $s,t_1$-cutsets are constructed: they are $\{a,b\}$, $\{a,e,c\}$ and $\{a,e,d\}$. These $s,t_1$-cutsets C can be inferred from the disconnected graphs $G - C$ at the leaves such as $G - a - b$. For each of these three disconnected graph $G - C$, COUNT_CO_STEINER_SUBSETS is invoked. Since $|F| = 1$, any subset of edges of the required size in $G - C$ is counted, for example, it returns three for $G - a - b$. After the edge contraction such as in $G - a - b$, a new $s,t_1$-cutset $\{e,c\}$ is constructed. The edges in this $s,t_1$-cutset are selected in the subsequent edge deletion-calls. Since we do not need to consider Steiner tree in this example, COUNT_CONTRACTED_CUTS is only computing the number of $s,t_1$-cutsets in $G$. In the next example (refer to Figure 7.5), we show the construction of $s,t$-cutsets $C$ that do not disconnect Steiner nodes, $|F| > 1$, in $G - C$.

*

COUNT_CONTRACTED_CUTS returns the number of $s,t$-cutsets of size $k$ in G for which there is a Steiner tree connecting the nodes in F
G is a graph
$s$ and $t$ are vertices of G
$F$ is a subset of the vertices of G such that $s \in F$, $t \notin F$, and the subgraph of $G$ induced by $F$ is connected
The following parameters apply to the graph $G \sim F$ where the supernode created by identifying the vertices in $F$ is labeled $s$
$\lambda_{st_{G \sim F}}$ is the minimum $s,t$-cutset size of G $\sim F$
$(S,\overline{S})$ is a minimum $s,t$-cutset of G $\sim F$

*\

COUNT_CONTRACTED_CUTS(G, s, t, F, $\lambda_{st_{G \sim F}}$, $k$, $(S,\overline{S})$)

1. [*Basis*]
   (a) If $\lambda_{st_{G \sim F}} = 0$, return ( COUNT_CO_STEINER_SUBSETS (G, F, $k$, T) ) where T is a Steiner tree that connects nodes in F
   (Note: there is always a Steiner tree T for $F$ because of the initial ordering of the vertices described in Section 7.2.1)
   (b) If $k < \lambda_{st_{G \sim F}}$ return(0)
   (c) { any set of k edges disconnects nodes in F }
   If ($| E(G) | -(| F | -1)) < k$, return(0)
2. Choose $f \in (S,\overline{S})$ of $G \sim F$ and let $e$ be the corresponding edge in G
3. [$G - e$]
   (a) n_delete = COUNT_CONTRACTED_CUTS(G-$e$,s,t,F, $\lambda_{st_{G \sim F}}$-1, $k$-1,$(S,\overline{S})$-{f})
4. [$G \sim e$]
   (a) if f $\neq (s,t)$
      i. Find the minimum $s,t$-cutset size of (G $\sim F$) f and denote it by $\lambda'$, and find $(S',\overline{S'})$, an $s,t$-cutset of (G $\sim F$) f of size $\lambda'$
      If $f = (u,s)$ or $f = (u,t)$, the supernode created by contracting $f$ is labeled $s$ or $t$ respectively
      ii. n_contract = COUNT_CONTRACTED_CUTS($G \sim e$, $s$, $t$, $F \sim e$, $\lambda'$, $k$, $(S',\overline{S'})$)
   (b) else n_contract = 0
5. return(n_delete + n_contract)

Table 7.2 COUNT_CONTRACTED_CUTS Algorithm

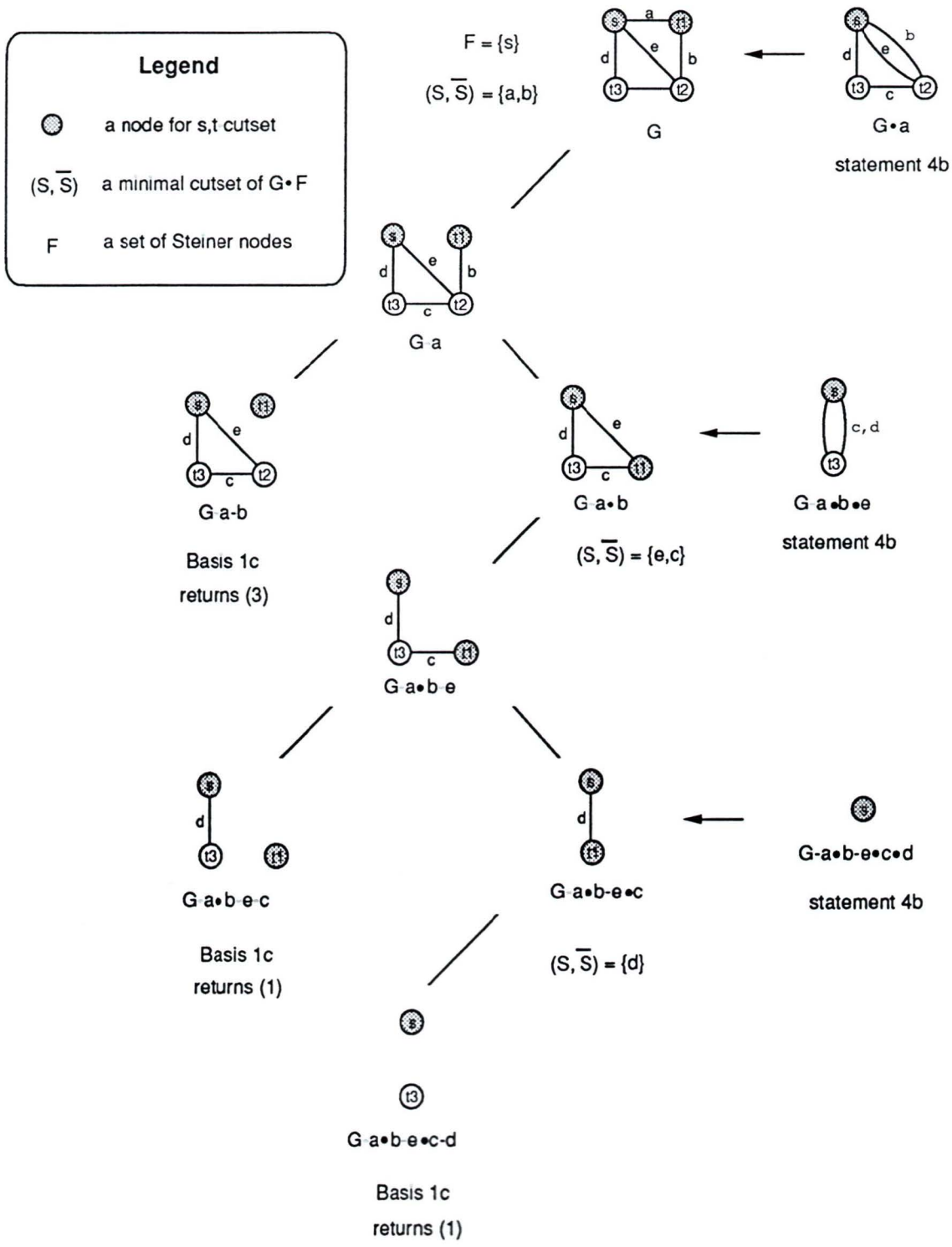



Figure 7 4 An Example for Algorithm COUNT_CONTRACTED_CUTS

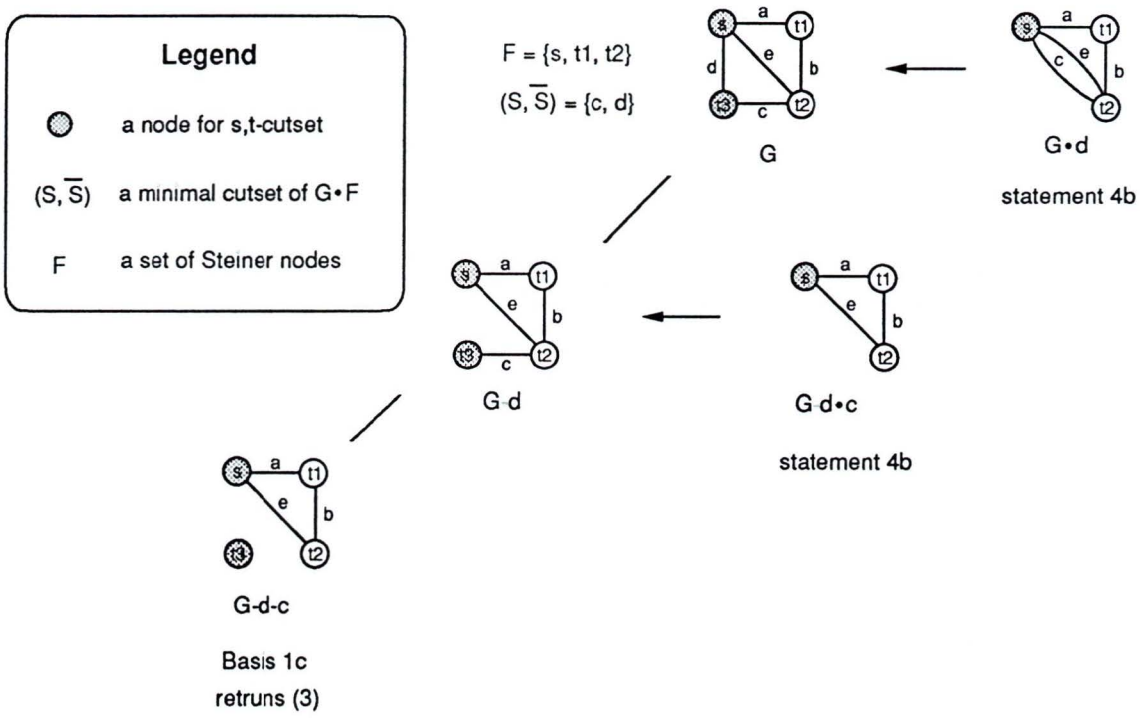



Figure 7 5 Example Two for Algorithm COUNT_CONTRACTED_CUTS

In Figure 7 5, $F = \{s, t_1, t_2\}$ and $(S,\overline{S}) = \{c, d\}$ COUNT_CONTRACTED_CUTS computes the number of $s, t_3$-cutsets $C$ such that in $G - C$, $s$ and $t_3$ are in different components and $s$, $t_1$ and $t_2$ are in the same component The $s, t_3$-cutset $\{c, d\}$ is constructed in ECA The edges in this $s, t_3$-cutset are selected in the subsequent edge deletion-calls Since the Steiner nodes $F$ in $G$ are $s$, $t_1$ and $t_2$, $t_2$ is $s$ in $G$ $F$ Thus, contracting $c = \{t_2, t_3\}$ in $G - d$ is the same as contracting $f = \{t_3, s\}$ in $(G - d)$ $F$ In this case, statement 4b is executed For the COUNT_CO_STEINER_SUBSETS call with $G - d - c$, the computation tree is in Figure 7 1

## Correctness of COUNT_CONTRACTED_CUTS

**Theorem 7 2 6** *COUNT_CONTRACTED_CUTS(G, s, t, F, $\lambda_{st_{G\,F}}$, k, $(S,\overline{S})$) counts cutsets C of size k such that in G-C, s and t are in different components and nodes*

*in $F$ are in the same component where*

1 *$(S,\overline{S})$ is a minimum $s,t$-cutset of $G - F$,*

2 *$\lambda_{st_{G-F}}$ is the $s,t$-edge connectivity of $G - F$,*

3 *$C$ is the set of edges in $E(G)$ that correspond to edges in $(S,\overline{S})$,*

4 *$F \subseteq V(G)$, $s \in F$, $t \notin F$, and*

5 *the subgraph of $G$ induced by $F$ is connected*

*These cutsets $C$ are referred as the qualified cutsets*

Proof The proof is developed according to whether $s$ and $t$ are connected in $G$ If $s$ and $t$ are not connected in $G$, $\lambda_{st_{G-F}}$ is zero Cutsets that satisfy the requirements in $G$ are co-Steiner subsets of $F$ in $G$ These cutsets are counted at Basis 1c by the call COUNT_CO_STEINER_SUBSETS$(G,F,k,T)$, where $T$ is a minimum Steiner tree for $F$ The minimum Steiner tree $T$ for $F$ always exists because of the labeling of vertices in $V(G)$ (refer to section 7 2 1 for details)

Suppose $s$ and $t$ are connected in $G$ Either one of the bases 1a or 1b is executed or no basis condition is satisfied If a basis condition is true, the return value is justified as follows Basis 1a is correct because of the following Lemma 7 2 4 guarantees that if $\lambda_{st_{G-F}} > k$, there is no $s,t$-cutset $C$ in $G$ such that nodes in $F$ are connected in $G - C$ and $|C| \leq k$ Basis 1b is correct because any Steiner tree for $F$ has at least $|F|$-1 edges

If $s$ and $t$ are connected in $G$ and no basis condition is satisfied, we use edge inclusion and exclusion, $G - e/G - e$, as in recurrence equation 3 1 1 for counting $s,t$-cutsets that satisfy in $G$ Statement 2 selects an edge $f$ in $(S,\overline{S})$ from $G - F$ Let $e$ in $G$ be the edge corresponding to $f$ By corollary 7 2 5, $e$ is in a qualified $s,t$-cutset of $G$ All qualified $s,t$-cutsets that use $e$ are counted at statement 3(a) by the call COUNT_CONTRACTED_CUTS$(G-e,s,t,F,\lambda_{st_{G-F}}-1,k-1,(S,\overline{S})-f)$ The invocation is correct because we are counting $s,t$-cutsets of size $k-1$ of $G-e$ and $(S,\overline{S})-f$ is a minimum $s,t$-cutset of $(G-e)-F$, that has size $\lambda_{st_{G-F}}-1$

All qualified $s,t$-cutsets that do not use $e$ are counted at statement 4 Two cases are possible First, when $f = (s,t)$, no $s,t$-cutset $C$ can be found in $(G \quad F) \quad f$, since $s$ and $t$ correspond to the same supernode Thus, statement 4(b) returns zero Second, when $f \neq (s,t)$, statement 4(a)i computes $\lambda'_{st_{G \; F}}$ and $(S',\overline{S'})$ of $(G \quad F) \quad f$ If $t$ is contracted into a node in $F$, $F \sim e$ represents the new Steiner node of $G$ Thus, the invocation, COUNT_CONTRACTED_CUTS(G $e$ s, t, $F \sim e$, $\lambda'_{st_{G \; F}}$, $k$, $(S',\overline{S'})$) at statement 4(a)ii, correctly counts qualified $s,t$-cutsets that do not use $e$

If $G$ is disconnected, basis 1(c) terminates the call Otherwise, we have to consider the following Every deletion-call and contraction-call decrements $|E(G)|$ by one Eventually, $|E(G)| = 0$ Since $|F| > 0$, if $|E(G)| = 0$, $|E(G)| - (|F| - 1) \leq 0$ Thus, if $k \geq 0$, when $|E(G)| = 0$, Basis 1b terminates the call Otherwise, $k < 0$ and Basis 1a terminates the call (by definition, $\lambda_{st_{G \; F}} \geq 0$) At least one of the basis conditions 1a, 1b and 1c is eventually true Thus, the algorithm always terminates

□

### 7.2.3 ECA

The equation 7 1 2 that counts cutsets of size $k$ in G is implemented by the ECA algorithm in Table 7 3

#### Example of ECA

An example illustrating how the algorithm ECA invokes COUNT_CONTRACTED_CUTS to count the number of cutsets of size three is shown in Figure 7 6 Given a graph $G$ that has four nodes and five edges ECA labels the nodes as $s$, $t_1$, $t_2$ and $t_3$, according to the vertex labeling in section 7 2 1 When $i = 1$, $F = \{s\}$ A minimum $s,t_1$-cutset $\{a,b\}$ is constructed When $i = 2$, $F = \{s,t_1\}$ A minimum $s,t_2$-cutset $\{b,c,e\}$ is constructed In this iteration, the minimum Steiner tree is $\{a\}$ When $i = 3$, $F = \{s,t_1,t_2\}$ A minimum $s,t_3$-cutset

*

ECA returns the number of cutsets of size $k$ in G
G is a connected graph
k is the size of the cutset being counted

*\

ECA(G,k)
Initially Label the nodes of G as $t_0, t_1, \ldots, t_{n-1}$ such that if $F_{i-1} = \{t_0, t_1, \ldots, t_{i-1}\}$ then $t_i$ is adjacent to some node in $F_i - 1$, for $i = 1, 2, \ldots, n-1$, by performing a depth first search starting any vertex $t_0$
Let $s = t_0$
$F_0 = \{s\}$
Let $N_k$ be the current sum of number of cutsets of size k computed so far
$N_k \leftarrow 0$

1 For i = 1 to n 1 do

(a) Find $\lambda_{st_i}(G \quad F_{i-1})$ the $s, t_i$-edge connectivity of $G \quad F_{i-1}$ and let $(S,\overline{S})$ be a minimum $s, t_i$-cutset of $G \quad F_{i-1}$

(b) $N_k = N_k +$ COUNT_CONTRACTED_CUTS(G, s, $t_i$, $F_{i-1}$, $\lambda_{st_i}(G \quad F_{i-1})$, k, $(S,\overline{S})$)

(c) $F_i = F_{i-1} \cup t_i$

2 return($N_k$)

Table 7 3 ECA Algorithm

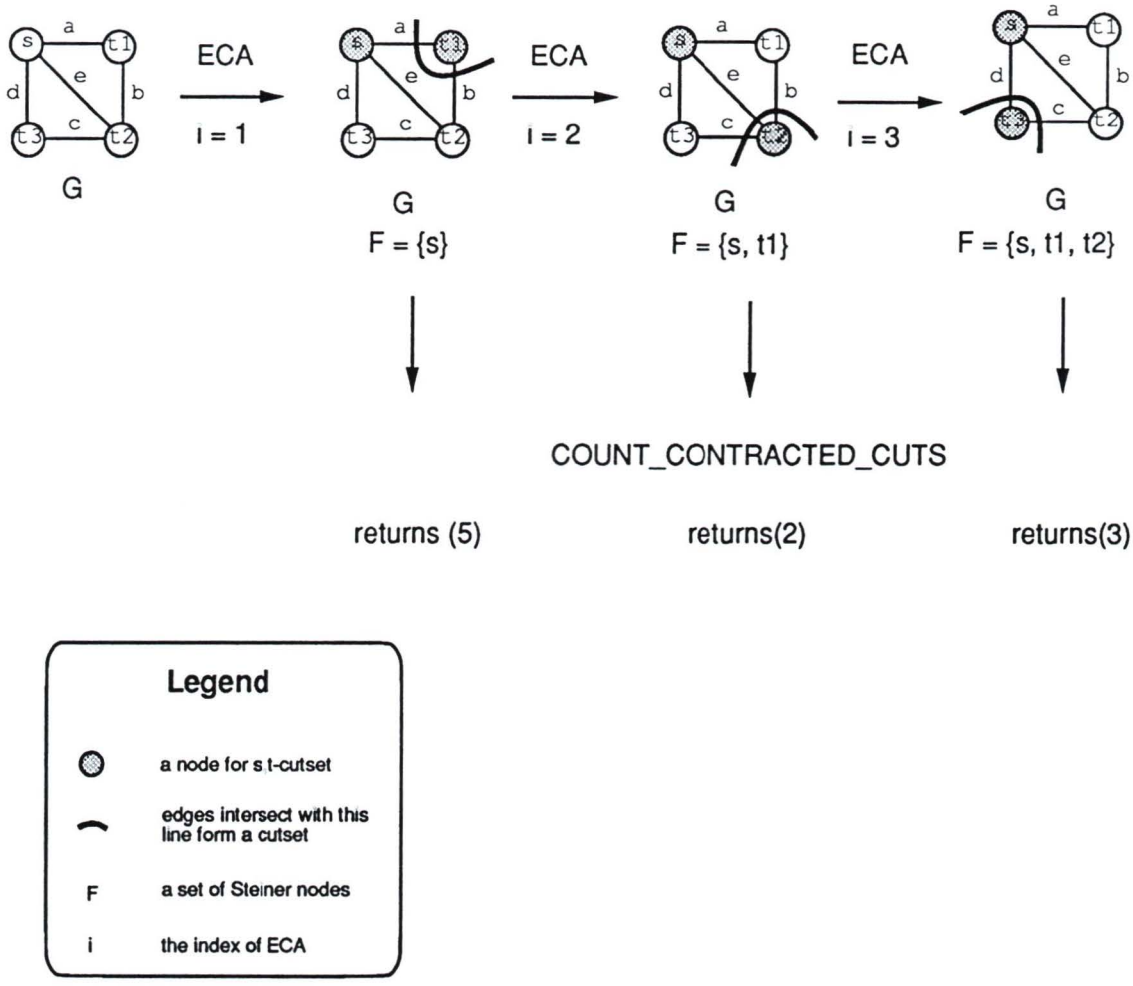



Figure 7.6: An Example for Algorithm ECA

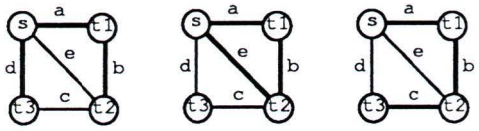
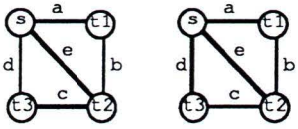
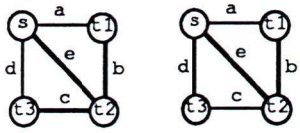
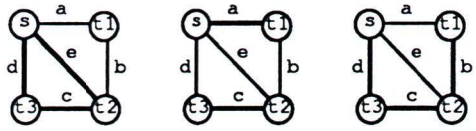
| ECA iteration | Co-Steiner Subsets Augmentation | **Cutsets Constructed (thick edges in a graph)** |
|---|---|---|
| i = 1 | Yes |  |
| | No |  |
| i = 2 | No |  |
| i = 3 | Yes |  |

Table 7.4 A Listing of the 3-edge Cutsets Constructed by ECA in Figure 7.6

$\{c, d\}$ is constructed. In this iteration, the minimum Steiner tree is $\{a, b\}$.

The 3-edge cutsets constructed by ECA is shown in Table 7.4. Cutsets that are constructed by ECA at iteration $i = 1$, 2 and 3 are grouped separately. At each iteration in ECA, cutsets that are constructed by augmenting co-Steiner subsets are grouped together. For example, at $i = 1$ in ECA a cutset $\{a, b\}$ is augmented with three co-Steiner subsets: they are $\{d\}$, $\{e\}$ and $\{c\}$. Examples of those cutsets that do not require co-Steiner subsets augmentation are $\{a, e, c\}$ and $\{a, e, d\}$ in ECA at iteration 1.

## 7.3 Complexity Analysis

ECA counts cutsets by invoking COUNT_CONTRACTED_CUTS $n-1$ times The dominant time complexity before each invocation is to compute an $s,t$-flow computation on $G$ $F_{i-1}$ That takes $A(k+1)$ time, because we can stop $s,t$-flow computation when more than $k$ augmenting paths are found Thus, ignoring the calls to COUNT_CONTRACTED_CUTS, ECA takes at most $nA(k+1)$ time In general, COUNT_CONTRACTED_CUTS constructs some $s,t$-cutsets for the input graph G For each $s,t$-cutset $C$, COUNT_CO_STEINER_SUBSETS is invoked with $G-C$ to construct all co-Steiner subsets

COUNT_CONTRACTED_CUTS counts the number of $s,t$-cutsets that satisfy the requirements using a $G-e/G$ $e$ recurrence equation Each non-terminating call invokes either two calls, one contraction and one deletion, or just one deletion-call when $s=t$ Statement 2 in table 7 2 always selects an edge from a qualified minimum $s,t$-cutset Thus, every internal call constructs at least a qualified $s,t$-cutset Since the number of $s,t$-cutsets constructed in COUNT_CONTRACTED_CUTS cannot be larger than $N_k$, the number of cutsets of size $k$, and every internal call returns non-zero value, we can conclude that COUNT_CONTRACTED_CUTS invokes at most $kN_k$ recursive calls by using similar arguments as in theorem 4 3 4 Ignoring a call to COUNT_CO_STEINER_SUBSETS, the dominant time complexity of a COUNT_CONTRACTED_CUTS call is the time required to update the maximum auxiliary graph for G $e$ or G-$e$ These updates take at most $(m+A(x+1))$ time (Theorem 5 1 3 and 5 1 4) Thus, ignoring the time in COUNT_CO_STEINER_SUBSETS, the computation time in COUNT_CONTRACTED_CUTS for finding all $s,t$-cutsets is $O((m+A(x+1))kN_k)$

Let $M(x)$ be the time complexity of COUNT_CO_STEINER_SUBSETS counting the number of co-Steiner subsets of size $x$

In the worst case, $\mid T \mid$ is $n-1$ A call to COUNT_CO_STEINER_SUBSETS for

co-Steiner subsets of size $x$ invokes $n-1$ more calls each which counts co-Steiner subsets of size $x-1$ (Statement 3 in Table 7.1). In each call, the dominant computation time is $O(m)$, that is for finding a new minimal Steiner tree at statement 3(a). Thus, $M(x) = O(m)(n-1)M(x-1)$. When $x$ is zero, COUNT_CO_STEINER_SUBSETS does not invoke other calls. Thus, $M(0) = 1$. Expanding the recurrence equation for $M(x)$, we get the following theorem.

**Theorem 7.3.1** $M(x) \in O(n^x m^x)$.

Each $s,t$-cutset of size $(\lambda + i)$ requires $M(x - i)$ computation time in COUNT_CO_STEINER_SUBSETS to construct cutsets of size $\lambda + x$. Thus in total, at most $O(\sum_{i=0}^{x} N_{\lambda+i} M(x-i))$ work is done by COUNT_CO_STEINER_SUBSETS.

$$\begin{aligned} O(\textstyle\sum_{i=0}^{x} N_{\lambda+i} M(x-i)) &\in O(\textstyle\sum_{i=0}^{x} n^{i+2} m^i M(x-i)) \\ &= O(\textstyle\sum_{i=0}^{x} n^{i+2} m^i n^{x-i} m^{x-i}) \\ &= O(\textstyle\sum_{i=0}^{x} n^{x+2} m^x) \\ &= O(x n^{x+2} m^x) \end{aligned}$$

Thus, the time complexity of the entire algorithm is
$O((m + A(x+1))kN_k + xn^{x+2}m^x + nA(k+1)) \in O((m + A(x+1))kn^{x+2}m^x)$.

**Theorem 7.3.2** *ECA takes $O((m+A(x+1))kn^{x+2}m^x)$ time to compute the number of cutsets of size $k$.*

We should note that the analysis is overestimating. Precise analysis is not obvious due to the variation of the number of recursive calls that depends on the selection of edges in a cutset.

# Chapter 8

# Summary

In summary, the time complexities of all the cutset counting algorithms are in Table 8.1. We also should be noted that the time complexity of BALL_PROVAN is $O(mn^2)$. ECA is slower than BALL_PROVAN in computing minimum cuts. However, ECA is at least factor $\Omega(n)$ faster than CR_NUM_CUTS and NUM_CUTS in the worst case. CR_NUM_CUTS and NUM_CUTS have the same worst case analysis. Since not every NUM_CUTS call performs $n-1$ $s,t$-flow computations, NUM_CUTS is faster than CR_NUM_CUTS. So far, the analysis have been overestimated for the *worst* cases. Similar techniques have been used to analyze all algorithms, so it is perhaps fair to compare them in the worst case.

| Algorithm | Table | Run-time Complexity |
|---|---|---|
| BASIC_NUM_CUTS | 3.1 | $O(m2^{k+n})$ |
| CR_NUM_CUTS | 4.1 | $2nA(k+1)kN_k = O(A(k+1)kn^{x+3}m^x)$ |
| NUM_CUTS | 5.1 | $2nA(k+1)kN_k = O(A(k+1)kn^{x+3}m^x)$ |
| ECA | 7.1 | $O((m+A(x+1))kn^{x+2}m^x)$ |

Table 8.1: Summary of Run-time Complexities

# Chapter 9

# Future Enhancements

The current ECA implementation suffers from one drawback. It is slower than Ball and Provan's algorithm when counting minimum cutsets.

When Ball and Provan enumerate cutsets, they only have to do $n$-1 $s,t$-flow computations. For each $s,t$-flow computation on $G$, a directed acyclic graph is constructed. They first define anti-chain in a directed acyclic graph $G'$ and then some theorems are stated to justify that the number of anti-chains in $G'$ is the same as the number of minimum $s,t$-cutsets in G. Ball and Provan's algorithm then counts the number of anti-chains in $G'$. Each anti-chain is derived in $O(m)$ time. We may be able to establish similar theorems with respect to minimal $s,t$-cutset and anti-chains. We could count anti-chains instead of edge inclusion and exclusion and as a result, flow computation for minimum cutset can be minimized. Thus, The time complexity of ECA would be the same as that of Ball and Provan's algorithm when counting minimum cuts, and faster for counting non-minimum cuts.

In ECA, an $s,t$-cutset $C$ is constructed in COUNT_CONTRACTED_CUTS and then augmented with co-Steiner subsets of size $x$. Given an $s,t$-cutset $C$ and the augmented co-Steiner subsets, we may be able to construct co-Steiner subsets for another $s,t$-cutset faster.

# Bibliography

[1] Charles J Colbourn, *The Combinatorics of Network Reliability*, New York, Oxford, Oxford University Press 1987

[2] J S Provan and M O Ball, "Calculating bounds on reachability and connectedness in stochastic networks", *Networks*, Vol 13 (1983) 253-278

[3] J S Provan and M O Ball, "The complexity of counting cuts and of computing the probability that a graph is connected", *SIAM Journal on Computing* 12 (1983) 777-788

[4] T B Brecht and C J Colbourn, "Lower bounds for two-terminal network reliability", *CCNG Report E-127*, University of Waterloo, 1985

[5] A Ramanathan and C J Colbourn, "Counting almost minimum cutsets with reliability applications", *Mathematical Programming* 39 (1987) 253-261

[6] Shimon Even, *Graph Algorithms*, Computer Science Press, 1979

[7] Ronald Gould, *Graph Theory*, The Benjamin/Cummings Publishing Company,Inc (1988) 75-76

[8] Gilles Brassard and Paul Bratley, *Algorithmics, Theory and Practice*, Prentice Hall, (1988) 132-133

[9] Ford, L R ,Jr and Fulkerson, D R , *Flows in Networks*, Princeton University Press, 1962

[10] M V Lomonosov and V P Polesskii, "Lower bounds of network reliability," *Problems of Information Transmission* 8 (1972) 118-123

# Vita

Surname Cheung | Given Names Kim Hung

Place of Birth Hong Kong | Date of Birth October 13, 1963

Educational Institutions Attended

| | |
|---|---|
| Simon Fraser University, Burnaby, Canada | 1984 to 1987 |
| University of Victoria, Canada | 1989 to 1991 |

Degrees Awarded

| | | |
|---|---|---|
| B Sc (Honours) | Simon Fraser University | 1987 |

Honours and Awards

| | |
|---|---|
| Hornour Roll, Simon Fraser University | 1985 |
| Simon Fraser University Open Scholarship | 1985-1986 |

Publications

Wendy Myrvold, Kim H Cheung, Lavon B Page and Jo Ellen Perry, Uniformly-Most Reliable Networks Do Not Always Exist, to be appeared in Networks

# Partial Copyright License

I hereby grant the right to lend my thesis (the title of which is shown below) to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying of this thesis for financial gain shall not be allowed without my written permission.

Title of Thesis:

## Counting Cutsets Faster

Author: [redacted]

Kim Hung Cheung

June 27, 1991