Graph Connectivity and Network Coding

LEUNG, Kai Man

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

The Chinese University of Hong Kong
August 2011

Abstract

Graph Connectivity and Network Coding

LEUNG, Kai Man

Master of Philosophy

Department of Computer Science and Engineering

The Chinese University of Hong Kong

2011

In this thesis we present a new algebraic formulation to compute edge connectivities in a directed graph, using the ideas developed in network coding. This reduces the problem of computing edge connectivities to solving systems of linear equations, thus allowing us to use tools in linear algebra to design new algorithms. Using the algebraic formulation we obtain faster algorithms for computing single source edge connectivities and all pairs edge connectivities, in some settings the amortized time to compute the edge connectivity for one pair is sublinear. Through this connection, we have also found an interesting use of expanders and superconcentrators to design fast algorithms for some graph connectivity problems.

摘要

Graph Connectivity and Network Coding

圖的連通度與網絡編碼

梁啓文

香港中文大學

計算機科學與工程學系

哲學碩士

二零一一

運用從網絡編碼中衍生的概念，我們在本論文提出一種新的代數公式來求算有向圖的邊連通度。此公式能把求算邊連通度的問題歸約爲求解多組線性方程，因而容許我們利用線性代數的工具來設計新的算法。我們運用此代數公式來獲得求算單源邊連通度及所有點對邊連通度的快速算法，其中在一些條件下計算一對頂點的邊連通度，其攤分時間複雜度爲次線性。透過當中的關係，我們亦發現有關擴展圖及超集中器的一些有趣應用，並用以設計一些有關解決圖的連通度問題的快速算法。

Acknowledgements

First and foremost, I would like to extend my deepest gratitude to my advisor Professor Lap Chi LAU, for his continuous support and invaluable advice. Without his countless help during these two years, my thesis is still far from complete.

I would also like to take this chance to express my heartfelt appreciation to all professors and students in the theory group for the enjoyable and fruitful days. I had a good time with all people who have been working in room 117. In particular, I must thank Leo Cheung, with whom I spent most of my time doing research and discussing on various interesting topics during these two years. His insightful ideas during our enjoyable collaboration are essential to establish this work.

Finally, I must thank my dear parents and brothers for always believing in me and their tremendous support. I also thank my beloved Yvette Yau, who brings me a colorful life since last summer. When I was lost and desperate, she is always there to comfort, advise and give me guidance. Her encouragement and moral support, and all the patience and understanding that she offered during the time of writing gave me the strength and faith to accomplish this thesis. I dedicate this thesis to them.

Contents

Chapter 1

Introduction

Graph connectivity is a basic concept that measures the reliability and efficiency of a graph. The *edge connectivity* of two vertices is the maximum number of edge disjoint paths from one vertex to another. Computing edge connectivities is a classical and well-studied problem in combinatorial optimization. Most known algorithms to solve this problem are based on network flow techniques.

In this thesis we present a new algebraic formulation to compute edge connectivities in a directed graph, using the ideas developed in network coding. Network coding is a new technique developed to solve the *multicasting* problem, where the source needs to send data to a set of receivers. The *multicasting rate* describes how much data can the source send to the receivers simultanouesly. The objective of the multicasting problem is to design a transmission scheme in order to maximize the multicasting rate. In the traditional routing method, data can only be received and forwarded. It is not optimal to use the traditional routing method for the multicasting problem. See Figure 1.1 for an example, where the multicasting rate is only one, using the traditional routing method.

Network coding is an innovative idea to overcome the inefficiency of traditional routing in the multicasting problem. The fundamental result of network coding [1] is the following: if the edge connectivity between the source to each receiver is at least k, then one can transmit k units of data to all receivers simultanouesly, by performing encoding and decoding at the vertices. See Figure 1.2 for an illustration, where the multicasting rate is two. Futhermore, it is shown that linear coding suffices for the multicasting problem [39]:

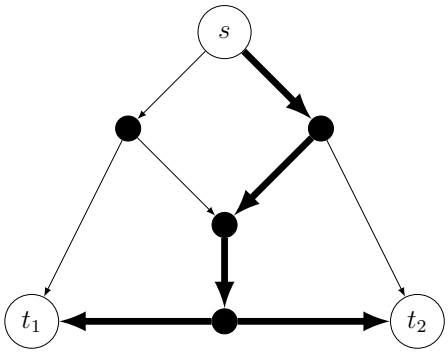


Figure 1.1: In the traditional routing method, when the source sends out data to both receivers simultaneously, the multicasting rate is only one.

To encode, the data on an outgoing edge of a vertex is a *linear combination* of the data on the incoming edges to the same vertex; to decode, any receiver can recover the original data by solving linear equations. Polynomial time algorithms have been developed to compute an optimal linear network coding scheme [39, 50, 45].

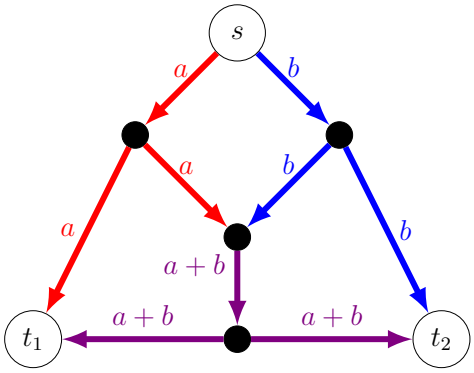


Figure 1.2: In this figure, the source sends out data a and b to the receivers. Encoding is done in the middle vertex, by performing addition of the incoming data. Each receiver can obtain the original data back by performing subtraction of its incoming data.

Our new algebraic formulation for computing edge connectivities is inspired by the random network coding algorithm [35] in constructing network codes. In the random network coding algorithm, the data on an outgoint edge of a vertex is a *random linear combination* of the data on the incoming edges to the same vertex. The advantage of this method is that one does not need to know the topology of the graph to compute an optimal linear coding scheme. It is previously known that in any directed acyclic graph, the edge connectivity can be computed by the *rank* of the incoming data to the receiver [38, 60]. An example is shown in Figure 1.3. We show that this algebraic

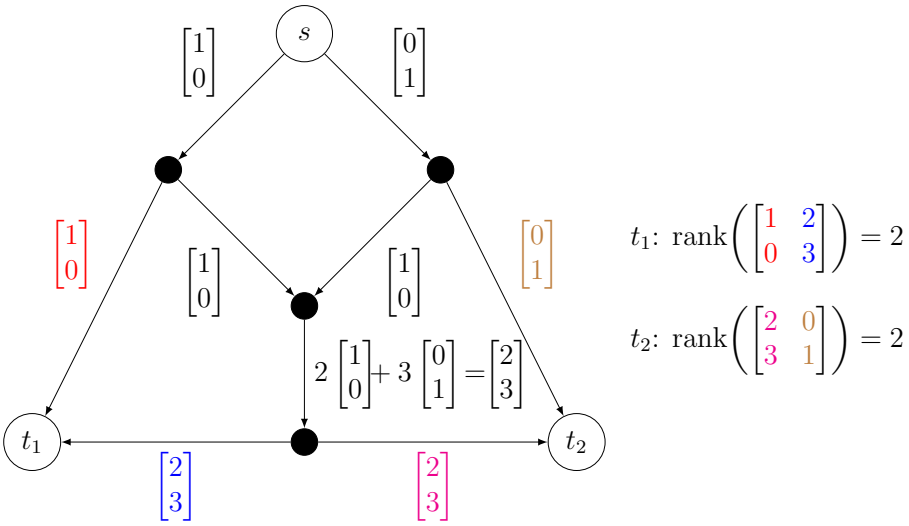


Figure 1.3: In this figure, the source sends out two unit vectors. The middle vertex sends out a random linear combination of its incoming data. The rank of the incoming data to each receiver is two, which matches their edge connectivities.

formulation can be extended to all directed graphs.

> *With random network coding, the edge connectivity of the source and the receiver in a directed graph is the rank of the incoming data to the receiver, with high probability.*

We will use this result to compute single source edge connectivities. In directed acyclic graph, we present a faster algorithm for computing edge connectivities by a simple transformation using the superconcentrator, which is a directed graph with high connectivities. To compute edge connectivities in some classes of graphs with constant maximum degree, such as planar graphs, bounded genus graphs and fixed minor free graphs, we can solve systems of linear equations by using a recent result of Alon and Yuster [5] based on the nested dissection method.

We also use our algebraic formulation to obtain faster algorithms for computing all pairs edge connectivities in general directed sparse graphs. Previously it is not known how to compute the edge connectivities faster than computing for each pair separately, even when the pairs share the source or the sink. We show that all pairs edge connectivities can be computed in one matrix inverse time, instead of solving the linear equations for each source vertex separately.

In the process we found an interesting use of superconcentrator and expander,

which are graphs that have strong connectivity properties. We discovered that both superconcentrator and expander can be used to design faster algorithms in other graph connectivity problems. We will show how to use them to improve the algorithms for finding edge splitting-off operations to preserve edge connectivities in directed and undirected graphs.

The results in this thesis are based on joint work with Ho Yee Cheung and Lap Chi Lau [15].

Chapter 2

Background

In this chapter we review some backgrounds on graph connectivity, network coding and algebraic algorithms. We first present some previous algorithms for the graph connectivity problems. Then we will introduce the history and the key ideas of network coding, which will be used to obtain the main results in this thesis. After that we will discuss some algebraic tools which help us to design faster algorithms.

2.1 Graph Connectivity

In this section we briefly describe the algorithms for the graph connectivity problems. We first discuss the algorithms for the edge connectivity problem and the vertex connectivity problem. Then we will present the main ideas for other graph connectivity problems, namely the all pairs edge connectivities problem and the edge splitting-off problem. After that we briefly introduce graph separators, the expanders, and the superconcentrators as the tools for us to obtain fast algorithms.

2.1.1 Preliminaries

We will introduce basic definitions and notations for graphs.

A directed graph $G = (V, E)$ consists of a set of vertices V and a set of edges E. An edge in E is an ordered pair of vertices $e = (u, v)$, and we say u is the *tail* of e and v is the *head* of e. An undirected graph $G = (V, E)$ is similar to directed graph, but we have

pairs in E being unordered. In an undirected graph, we say u and v are the *endpoints* of the edge $e = (u, v)$, and e is *incident* to both u and v. Unless otherwise specified, we assume, throughout the whole thesis, that $|V| = n$ and $|E| = m$.

We introduce some special classes of graphs that we will use in this thesis. A graph is *simple* if there are no parallel edges. An undirected graph is called a *tree* if $m = n-1$ and for every pair of vertices $u, v \in V$ there is exactly one path connecting u and v. A graph is *planar* if it can be drawn in a two dimensional plane so that the edges only intersect at the vertices. The *bounded genus graph* is a graph such that it can be embedded in a surface of constant genus without any of its edges crossing one another. Informally, the *genus* is an orientable surface, which can be obtained by attaching finitely many "handles" to the sphere. For example, a planar graph has genus zero. A graph G' is a *minor* of a graph G if G' can be obtained from a subgraph of G by contracting edges, deleting some edges and deleting some isolated vertices. A graph G is *fixed minor free* if a fixed graph H is not a minor of G. For example, a planar graph is K_5-minor free and $K_{3,3}$-minor free. The *line graph* L_G of a directed graph G is a directed graph with m vertices such that each vertex in L_G represents an edge in G, and an edge (u, v) is in L_G if u and v correspond to the edges e_u and e_v in G so that the head of e_u is the tail of e_v.

In this paragraph we shall define the vertex cuts of a graph. In a directed graph, we define $\delta^{in}(v) = \{(u, v) : (u, v) \in E\}$ as the set of incoming edges of v; similarly we define $\delta^{out}(v) = \{(v, u) : (v, u) \in E\}$ as the set of outgoing edges of v. In addition, we define $\delta^{in}(S) = \{(u, v) : u \notin S, v \in S \text{ and } (u, v) \in E\}$ as the set of incoming edges of $S \subseteq V$. Similarly we also have $\delta^{out}(S) = \{(u, v) : u \in S, v \notin S \text{ and } (u, v) \in E\}$. The indegree of a vertex v is defined as $d^{in}(v) = |\delta^{in}(v)|$. Similarly we define the outdegree of v as $d^{out}(v) = |\delta^{out}(v)|$. We have similar definitions of indegree and outdegree of a subset $S \subseteq V$, which are defined as $d^{in}(S) = |\delta^{in}(S)|$ and $d^{out}(S) = |\delta^{out}(S)|$, respectively. In an undirected graph $G = (V, E)$, we define $\delta(v) = \{(u, v) : (u, v) \in E\}$, $d(v) = |\delta(v)|$, and $\delta(S) = \{(u, v) : (u, v) \in E, u \in S, v \in V - S\}$ accordingly.

2.1.2 Edge Connectivity

The notion of s-t edge connectivity is defined as a measure on how well two vertices s and t are connected. We denote $\lambda_{s,t}$ as the s-t edge connectivity. There are two definitions.

- *Edge-disjoint path:* We define the edge connectivity between s and t, or the *s-t edge connectivity* in short, as the maximum number of edge disjoint paths from s to t.
- *Cut size:* A subset of edges $\delta^{out}(U) \subseteq E$ is called a *cut* for some subset of vertices $U \subseteq V$. If $s \in U$ and $t \notin U$, then $\delta^{out}(U)$ is called an s-t cut. The s-t edge connectivity can also be defined as the smallest size of an s-t cut, which is the minimum cardinality of $\delta^{out}(U) \subseteq E$ for some $U \subseteq V$ such that $s \in U$ and $t \notin U$.

Menger's Theorem states that the above definitions are equivalent. See Figure 2.1 for an example.

Theorem 2.1 (Menger's Theorem [54]). *Let $G = (V, E)$ be a directed graph, and $s, t \in V$. Then the maximum number of edge-disjoint s-t paths is equal to the minimum size of an s-t cut.*

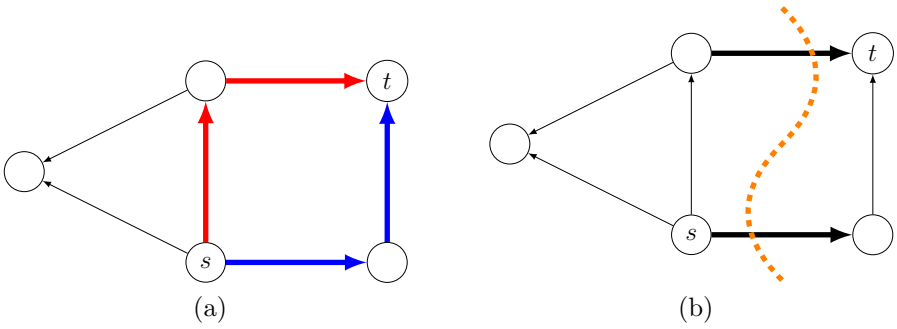


Figure 2.1: Figure 2.1a shows the maximum of two edge-disjoint paths, which are indicated by thick edges. Figure 2.1b shows a minimum s-t cut with two edges. In this example, the maximum number of edge-disjoint paths equals to the minimum number of s-t cut.

2.1.3 Vertex Connectivity

In this section, we will first define the s-t vertex connectivity in directed graphs, and show that determining the s-t vertex connectivity can be reduced to the bipartite matching problem.

The s-t vertex connectivity is the maximum number of s-t vertex disjoint paths, which are s-t paths that do not share vertices except s and t. The s-t vertex connectivity problem can be reduced to the s-t edge connectivity problem. Given a directed graph G and two distinct vertices s and t, we split every vertex $v \in V - s - t$ to v' and v'', and add an edge (v', v''). Every incoming edge of v is connected to v' and every outgoing edge of v is connected from v''. Let G' be the transformed graph as described, then the s-t vertex connectivity in G is equal to the s-t edge connectivity in G'. An example is illustrated in Figure 2.2.

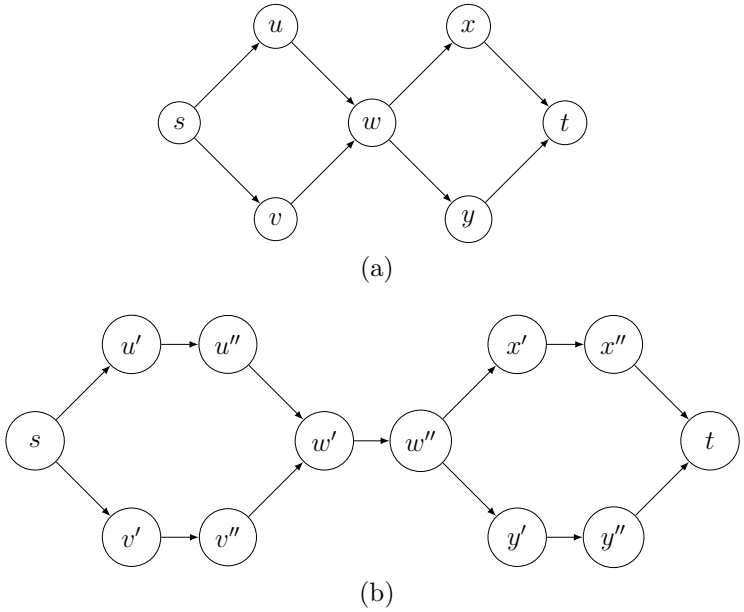


Figure 2.2: An example graph G is shown in Figure 2.2a. Figure 2.2b shows the resulting graph G' after the transformation. It can be seen that s-t vertex connectivity in G equals to s-t edge connectivity in G'.

The s-t vertex connectivity problem can also be reduced to the bipartite matching problem. An undirected graph $G = (L, R, E)$ is a bipartite graph with two disjoint sets of vertices L and R such that every edge connects a vertex in L to a vertex in R. A subset $M \subseteq E$ of edges is a matching if the edges in M are vertex disjoint. The bipartite matching problem is to find a maximum cardinality matching in a bipartite graph. Figure 2.3 shows an example bipartite graph with maximum matching of size three. The main idea of the reduction is presented in [58](Section 16.7c), and we briefly describe it here. We assume that s and t have no common neighbors, otherwise we can safely remove such vertices and this will decrease the vertex connectivity exactly by one.

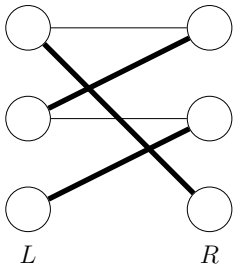


Figure 2.3: An example bipartite graph with three vertices on both left and right hand sides. The maximum matching is shown by thick edges.

Let S be the set of neighbors from s, and T be the set of neighbors to t. We construct the bipartite graph $G' = (L', R', E')$ as follows. First we split each vertex $v \in V - S - T$ in G to two vertices v_{in} and v_{out}, place them in L' and R' respectively, and add an edge $(v_{\text{out}}, v_{\text{in}})$. For every vertex $v \in S$, we place a vertex v_{out} in L'. For every vertex $v \in T$, we place a vertex v_{in} in R'. An edge $(u_{\text{out}}, v_{\text{in}})$ is added if u is in $V - T$ and v is in $V - S$. See Figure 2.4 for an example. Using the edges $(v_{\text{out}}, v_{\text{in}})$, there is a bipartite matching of size $n - |S| - |T|$. Observe that each s-t vertex disjoint path increases the size of this bipartite matching in G' by one. Thus there are k vertex disjoint paths from s to t in G if and only if there is a bipartite matching of size $n - |S| - |T| + k$ in G'.

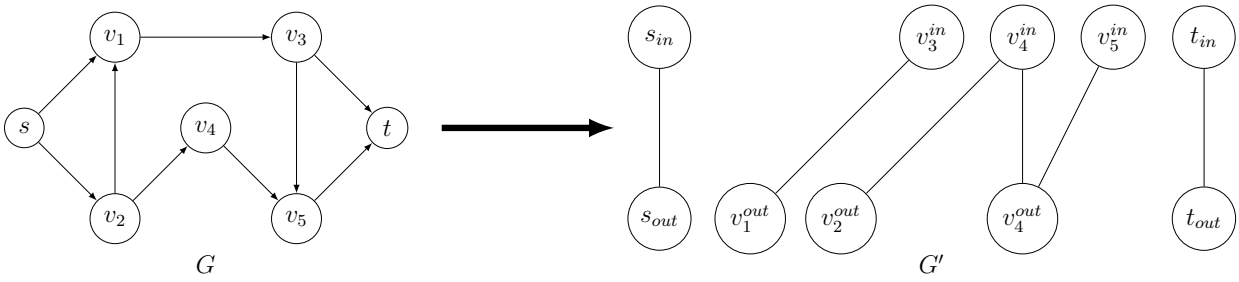


Figure 2.4: An example of reducing the s-t vertex connectivity problem to the bipartite matching problem.

2.1.4 Algorithms for Graph Connectivities

Most known algorithms to compute the s-t edge connectivity are based on network flow techniques. In the case G is a simple and uncapacitated graph, the fastest known algorithm that computes $\lambda_{s,t}$ is given by Even and Tarjan [19]. Their algorithm runs in $O(\min\{\sqrt{m}, n^{2/3}\} \cdot m)$ time. In Section 2.1.3, it is discussed that the s-t vertex

connectivity problem can be reduced to the s-t edge connectivity problem. Therefore vertex connectivity can also be solved in the same running time. In the worst case the algorithm runs in $O(n^{\frac{8}{3}})$ time. For the bipartite matching problem, the fastest known combinatorial algorithm is given by Hopcroft and Karp [37]. Their algorithm computes a maximum bipartite matching in $O(m\sqrt{n})$ time. When $m = O(n^2)$, their algorithm runs in $O(n^{2.5})$ time. The fastest known algebraic algorithm runs in $O(n^{\omega})$ time [32, 55], which is faster than combinatorial algorithm since $\omega \approx 2.376$ [11]. As mentioned in Section 2.1.3, the s-t vertex connectivity can also be computed in the same running time.

The s-t edge connectivity problem can also be reduced to the s-t vertex connectivity problem, but it is less efficient. The known reduction is based on the line graph L_G (see Section 2.1.3 for the definition). Let $X = \delta^{out}(s)$ and $Y = \delta^{in}(t)$. Add two new vertices s' and t' in L_G. First connect s' to each vertex in L_G which corresponds to an edge in X. Then connect each vertex in L_G that corresponds to an edge in Y to t'. The s'-t' vertex connectivity in L_G is equal to s-t edge connectivity in G. Recall that the s-t vertex connectivity problem can be solved by bipartite matching (see Section 2.1.3 for the reduction). So we can reduce the s-t edge connectivity problem to the bipartite matching problem. However we need to find a maximum bipartite matching in a graph with $O(m)$ vertices, and the resulting algorithm is slower than the fastest known algorithm.

2.1.5 All Pairs Edge Connectivities

To compute all pairs edge connectivities in directed graph, no previous known result is faster than $O(n^2)$ computations of single pair edge connectivity. In undirected graph there is an efficient method to compute all pairs edge connectivities by constructing a Gomory-Hu tree, which is a succint data structure to obtain all pairs edge connectivities for any undirected graph.

A Gomory-Hu tree is a tree $T = (V, F)$. Every edge in F has a value. For every pair of vertices s and t, there is a unique s-t path in T. The minimum value among the edges in the s-t path is the s-t edge connectivity. Hence T is a compact representation of all pairs edge connectivities in G. See Figure 2.5 for an example.

Gomory and Hu [30] showed that a Gomory-Hu tree can be constructed by $n-1$ edge

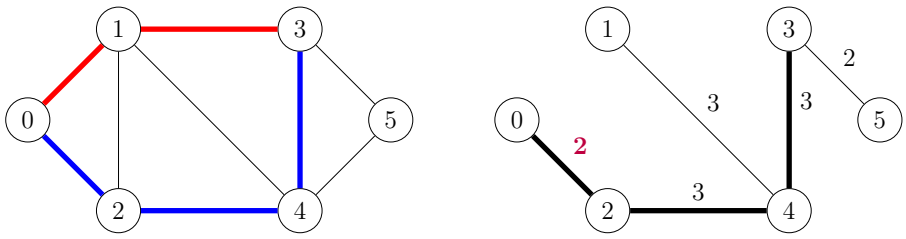


Figure 2.5: An undirected graph G is shown in Figure 2.5a. Its Gomory-Hu tree T is shown in Figure 2.5b. T represents all pairs edge connectivities of G. For example, the edge connectivity between vertex 0 and vertex 3 in G is two, which is shown by thick edges. The path connecting them in T is shown by thick edges. We can obtain the edge connectivity by the edge with smallest value along the path, which is shown in bold.

connectivity computations. Recently Bhalgat, Hariharan, Kavitha and Panigrahi [8] gave the fastest $\tilde{O}(mn)$ Gomory-Hu tree construction algorithm, where $\tilde{O}(f) = O(f \text{ polylog } f)$. This algorithm can be used in Section 2.1.6 to obtain fast algorithms in the edge splitting-off problem in undirected graphs.

2.1.6 Edge Splitting-off

In this section, we will introduce the edge splitting-off problem. Then we will mention its applications, and present previous results on the problem.

Splitting-off a pair of edges (ux, xv) means deleting these two edges and adding a new edge (u, v) if $u \neq v$. Note that the above definition works for both undirected and directed graphs. The content of the edge splitting off theorems is to prove the existence of one pair of edges (ux, xv) so that its splitting-off preserves the edge connectivities for all pairs of vertices. An example is described in Figure 2.6.

In undirected graph, Mader [53] proved that there is a "good" pair of edges in almost all situations.

Theorem 2.2 (Mader [53]). *Let $G = (V, E)$ be an undirected graph and $x \in V$. If there is no cut edge incident to x and $d(x) \neq 3$, then there exists an edge pair (yx, xz) so that its splitting-off preserves the edge connectivity for every pair of vertices $a, b \in V - x$.*

There is a similar theorem for Eulerian directed graphs where for every vertex its

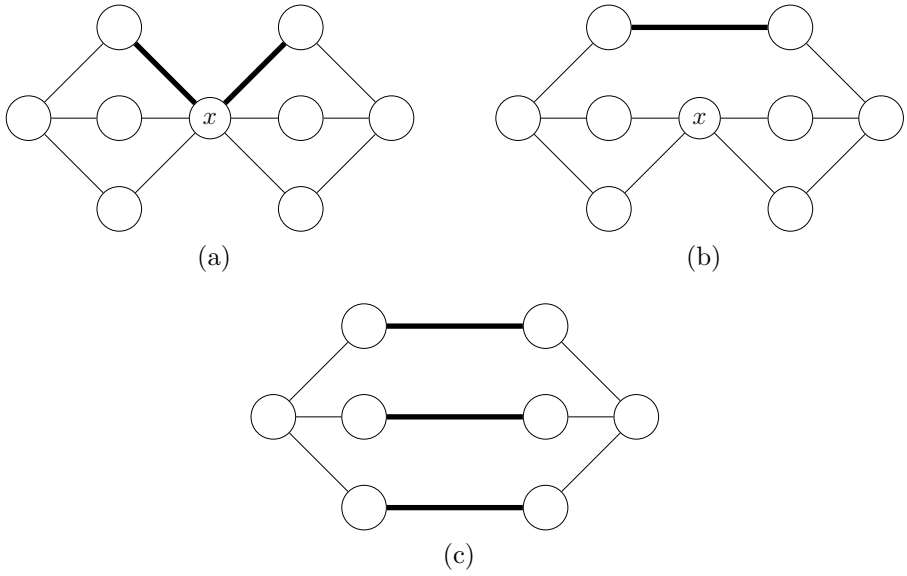


Figure 2.6: An undirected graph G is shown in Figure 2.6a. Splitting-off the pair of thick edges results in Figure 2.6b. Figure 2.6c shows the result when x is completely split-off.

indegree is equal to its outdegree.

Theorem 2.3 (Bang-Jensen, Frank and Jackson [6])**.** *Let $G = (V, E)$ be an Eulerian directed graph and $x \in V$. Then there exists an edge pair (yx, xz) so that its splitting-off preserves the edge connectivity for every ordered pair of vertices $a, b \in V - x$.*

These results are a powerful tool for proving theorems and developing algorithms for many graph connectivity problems, including connectivity augmentation problems [20, 13, 7], network design problems [28, 42, 12], tree packing problems [6, 47, 9] and graph orientation problems [21].

In undirected graphs, when $d(x)$ is even, Mader's theorem can be repeatedly applied until x is of degree zero. In directed graphs, Theorem 2.3 can also be repeatedly applied until x is of degree zero. We call this a *complete splitting-off* at x. See Figure 2.6c for an example. In the remaining of the section we will discuss previous results on completely splitting-off vertices in undirected and directed graph respectively.

In undirected graph, to completely split-off a vertex x, the simplest way is to try all $O(d(x)^2)$ pairs, and then check whether the edge connectivities are preserved. We can construct a Gomory-Hu tree (as discussed in Section 2.1.5) to efficiently do the checking. Lau and Yung [48] proved that $O(d(x))$ attempts suffices to completely split-

off x, using structural theorems of mincuts. They also obtained faster algorithms to check whether connectivities are preserved, using the fast Gomory-Hu tree algorithm by Bhalgat, Hariharan, Kavitha and Panigrahi [8]. We note that there is an earlier algorithm proposed by Gabow [23] for the complete splitting-off problem in undirected graphs.

In directed graph, there are no known results that can completely split-off a vertex x faster than the straightforward algorithm.

2.1.7 Graph Separator

In this section, we will describe the concept of graph separators, whose removal breaks an undirected graph into components such that all components are not too large. Then we shall present previous results of graph separators on planar graphs and other classes of graphs. In Section 2.3.2 we will discuss an application of graph separators to solve systems of linear equations.

Definition. *Let $G = (V, E)$ be an undirected graph, and $f : \mathbb{N} \to \mathbb{R}$ be a function, $\alpha \in (0, 1)$. Then $Z \subseteq V$ is a $(f(n), \alpha)$-separator if $|Z| \leq f(n)$ and $V - Z$ can be further partitioned into two parts X and Y such that $|X \cup Z| \leq \alpha n$, $|Y \cup Z| \leq \alpha n$, and no edges have endpoints in both X and Y. We also say that G has a $(f(n), \alpha)$-separation.*

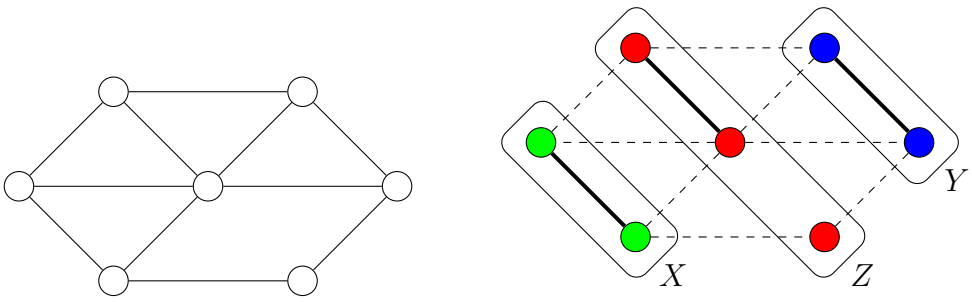
For example, if G is a tree, then G has $(1, 1/2)$-separation, which can be found in $O(n)$ time [41]. A class of graphs is *hereditary* if it is closed under taking subgraphs, meaning that for any graph G in the class, any subgraph G' of G also belongs to the same class. For example, the classes of planar graphs, bounded genus graphs and fixed minor free graphs are hereditary. If every graph in a hereditary class has a $(f(n), \alpha)$-separation, then we can recursively apply the same argument to the subgraphs induced by X and Y respectively, until X and Y are of constant size. We can represent such process by a tree structure called a *weak separator tree*. For completeness, we present the formal definition in [5].

Definition. *For any undirected graph $G = (V, E)$ such that it belongs to a hereditary class of graphs and has a $(f(n), \alpha)$-separation, a tree $T = (V', F)$ is called a weak separator tree if the following properties are satisfied:*

- *Every vertex* $u \in V'$ *of* T *is associated with a vertex subset* $V_u \subset V$ *of* G.
- $\bigcup_{u \in V'} V_u = V$. *For every* $u, u' \in V'$ *such that* $u \neq u'$, $V_u \cap V_{u'} = \emptyset$.
- *Every vertex in* T *either has two children or itself is a leaf.*
- $|V_u| = O(1)$ *for each leaf* u *in* T. *Otherwise, let* u_1 *and* u_2 *be its children. Then* V_u *is the* $(f(n'), \alpha)$*-separator of the subgraph of* G *induced by* $V_u \cup V_{u_1} \cup V_{u_2}$ *where* $|V_u \cup V_{u_1} \cup V_{u_2}| = n'$, *and* V_{u_1} *and* V_{u_2} *are the separated components.*

For planar graphs, there is a nice separator theorem proven by Lipton and Tarjan [52]. An example is illustrated in Figure 2.7.

Theorem 2.4 ([52]). *Any undirected planar graph has a* $(O(\sqrt{n}), 2/3)$*-separation, which can be found in* $O(n)$ *time.*



(a) (b)

Figure 2.7: Figure 2.7a shows an example undirected graph. Its separator Z, and its two separated components X and Y are shown in Figure 2.7b.

There are other hereditary classes of graphs which also have a $(O(\sqrt{n}), \alpha)$-separation. For the class of bounded genus graph a $(O(\sqrt{n}), \alpha)$-separator can be found in linear time [26]. For the class of fixed minor free graph, there is an $O(n^{1.5})$ algorithm [4] to find a $(O(\sqrt{n}), \alpha)$-separator.

The notion of a weak separator tree will be used in Section 2.3.2 to develop algorithms in solving the system of linear equations.

2.1.8 Expander Graphs

An expander graph $G = (V, E)$ is a sparse undirected graph that exhibits strong connectivity properties.

Definition. *A graph $G = (V, E)$ is called an (n, d, c)-expander if it has n vertices, the maximum degree is d, and for all $S \subset V$ with $|S| \leq |V|/2$, we have $d(S) \geq c|S|$ and c is called the edge expansion of G.*

The abundance of expander graphs was discovered by Pinsker [56]. He proved by probabilistic method that a sparse random graph is an expander, with high probability.

Theorem 2.5 ([56])**.** *There exist a constant $c_0 > 0$ such that for any constant $d \geq 3$ and any even integer n, there is an (n, d, c_0)-expander.*

Theorem 2.5 is not constructive, and in many applications a deterministic construction algorithm is needed. We refer the readers to the excellent survey by Hoory, Linial and Wigderson [36] about many known constructions in details.

We present a well known deterministic construction ([22] and [36], Section 2.2) of regular expander graph H with n vertices for our application in Section 3.6. Let p be the smallest prime number greater than n. It is known that $p \leq 2n$. Then we construct $H = (V', E')$ with $|V'| = p$ and we label its vertices with $\{0, 1, \cdots, p-1\}$. For the i-th vertex where $i > 0$, it has exactly three neighbors, namely $(i+1) \bmod p$, $(i-1) \bmod p$ and $i^{-1} \bmod p$, where $i^{-1} \bmod p$ is the multiplicative inverse of $i \bmod p$. Note that it is possible to have self loops. For the vertex labeled 0, its neighbors are $0, 1$ and $p-1$. Figure 2.8 illustrates the result of H when $p = 7$.

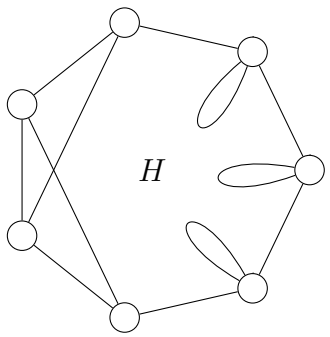


Figure 2.8: An example expander H with 7 vertices.

For our application in Section 3.6 we need a graph with $d(S) \geq |S|$ for all S with $|S| \leq |V|/2$. We show that H^8 will satisfy this property, by using the concept of conductance and its relation to the second largest eigenvalue of the adjacency matrix.

Theorem 2.6 ([17, 3, 2], Theorem 2.4 of [36]). *Let $G = (V, E)$ be an n-vertex undirected d-regular graph with its adjacency matrix $A(G)$. Let $\lambda_1, \lambda_2, \cdots, \lambda_n$ be the eigenvalues of the matrix $\frac{1}{d}A(G)$ such that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. For $S \subseteq V$ with $|S| \leq |V|/2$, we define*

$$h(S) = \frac{d(S)}{d \cdot |S|},$$

and we define

$$h(G) = \min_{0 \leq |S| \leq \frac{|V|}{2}} h(S).$$

Then

$$h(G) \geq \frac{1 - \lambda_2}{2}.$$

To ensure that $d(S) \geq |S|$ for every $|S| \leq |V|/2$, it suffices to show that $h(G) \geq \frac{|S|}{d \cdot |S|} = \frac{1}{d}$. By Theorem 2.6, if we can construct a d-regular expander graph H' such that

$$\lambda_2 \leq 1 - \frac{2}{d}, \tag{2.1}$$

then we can deduce that $h(H') \geq \frac{1-\lambda_2}{2} \geq \frac{1}{d}$. It is proved that H is a good expander.

Theorem 2.7 ([59]). *The second largest eigenvalue of the normalized adjacency matrix of H is strictly less than $1 - 1/10^4 = 0.9999$.*

To construct a graph with $\lambda_2 \leq 1 - \frac{2}{d}$, we can take the k-th graph powers of H to obtain a new graph H' such that it is a 3^k-regular graph with the second largest eigenvalue being λ_2^k. One can verify that if we choose $k = 8$, the inequality (2.1) will be satisfied for H'. Since we take constant graph power of H, the vertex degree in H' is still a constant (although much larger) and the construction time is $O(n \log n)$. Therefore H' is a $(p, 3^8, 1)$-expander. We will use this construction to obtain fast algorithms for the complete edge splitting-off problem in Section 3.6.

2.1.9 Superconcentrator

Superconcentrator is a sparse directed acyclic graph with strong connectivity properties. It is first defined by Valiant [61] for studying the complexity of linear transformations.

Definition ([36]). *Let $G = (V, E)$ be a directed graph and let I and O be two subsets of V with n vertices. We say that G is a superconcentrator if for every k and every $S \subseteq I$ and $T \subseteq O$ with $|S| = |T| = k$, there exist k vertex disjoint paths from S to T.*

The existence of superconcentrator with $O(n)$ edges is proved by Valiant. There exist explicit constructions [22, 36] of superconcentrators with the following properties:

1. There are $O(n)$ vertices and $O(n)$ edges,

2. the maximum indegree and the maximum outdegree are constants, and

3. the graphs are directed acyclic.

We also show the construction of superconcentrator in [22, 36] with n inputs and n outputs in the remaining of this section. The construction is recursive. The base case is when n is a small constant n_0. Then we just build a bipartite graph with n_0 vertices on both sides, and add $O(n_0^2)$ directed edges (u, v) for any vertex u on the left side and v on the right side. To construct a superconcentrator with $n > n_0$ inputs and n outputs, we need three graphs. Let G_1 be a bipartite constant degree expander graph with n vertices on the left and δn vertices on the right, where δ is a constant less than one. It can be constructed in $O(n)$ time [22]. See Figure 2.9b for illustration. Let G_2 be a bipartite constant degree expander graph with δn vertices on the left and n vertices on the right. We connect the vertices in G_2 similar to G_1 with directions reversed, as depicted in Figure 2.9c. Let C be a superconcentrator with δn inputs and δn outputs, which exists by inductive assumption. Then we obtain our desired superconcentrator by the following procedures:

1. Connect edges from each vertex on right side of G_1 to a distinct input of C.

2. Connect edges from each output of C to a distinct vertex on the left side of G_2.

3. Connect edges from each vertex on left side of G_1 to a distinct vertex on the right side of G_2.

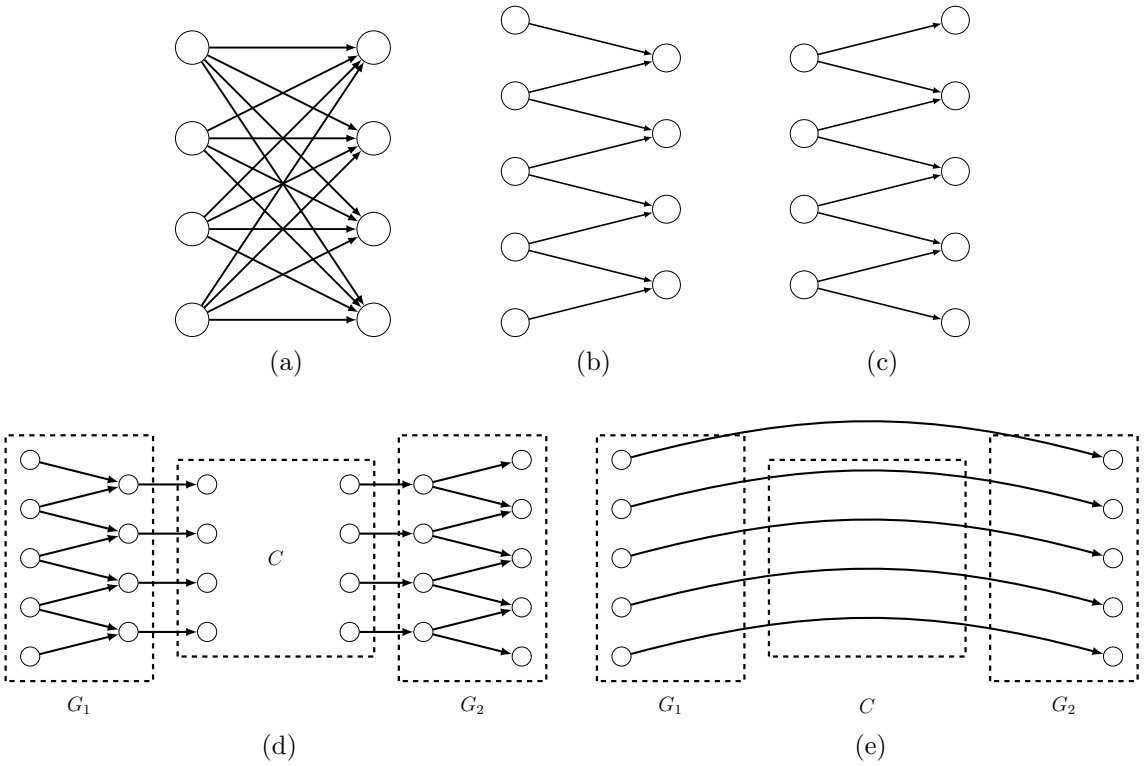


Figure 2.9: An illustration of recursive construction of superconcentrator. Figure 2.9a shows the base case when $n_0 = 4$, in which it is a bipartite graph with 16 edges. To construct a superconcentrator with $n > n_0$ inputs and outputs, we need graphs G_1 and G_2, which are shown in Figure 2.9b and Figure 2.9c respectively. The connection steps are shown in Figure 2.9d and Figure 2.9e separately. Figure 2.9d shows the connection between G_1 and C, and between C and G_2, where C is a superconcentrator of smaller size. We do not expose the connections within C. Figure 2.9e shows the last connection steps. Combining, the whole graph is a superconcentrator with n inputs and outputs that satisfies the desired properties.

See Figure 2.9 for an illustration. It is proved in [22, 36] that the above construction of superconcentrator satisfies the properties.

We will state the running time of the above construction. Let $T(n)$ be the running time of the construction of superconcentrators with n inputs and n outputs. Then we can obtain the following recurrence relations

$$T(n) = T(\delta n) + O(n).$$

It follows that $T(n) = O(n)$. Hence the construction can be implemented in $O(n)$ time.

2.2 Network Coding

2.2.1 Concept

In this section we will first define the multicasting problem. Then we will review previous work on the multicasting problem by the traditional routing method. After that we present the idea of network coding and show its advantage over routing.

The multicasting problem is the following: Given a directed acyclic graph $G = (V, E)$ with a source vertex s and a set $T \subset V$ of receivers, we want to transmit data from the source s to the receivers in T. Each edge $e \in E$ carries one unit of data. The *multicasting rate* is h if every receiver can receive h units of data in a transmission. The objective of the multicasting problem is to maximize h.

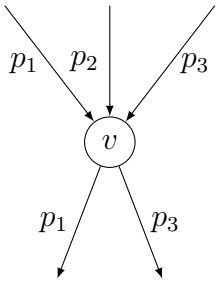


Figure 2.10: An example of routing inside a vertex v. In this figure, v has three incoming edges and two outgoing edges. The data p_1 and p_2 are routed to the left and right outgoing edges respectively. Note that p_2 is not routed.

Routing is the classical technique for the multicasting problem. In routing, the data on an outgoing edge of a vertex v is just a *copy* of the data on some incoming edge to v. See Figure 2.10 for illustration. Achieving the optimal rate by routing is shown to be equivalent to finding an optimal Steiner tree packing. However the problem is shown to be NP-hard [40].

Network coding is a novel technique to increase the multicasting rate in the multicasting problem. The study of network coding was first initiated by Ahlswede, Cai, Li and Yeung [1]. In network coding, each vertex is allowed to *encode* and *decode*

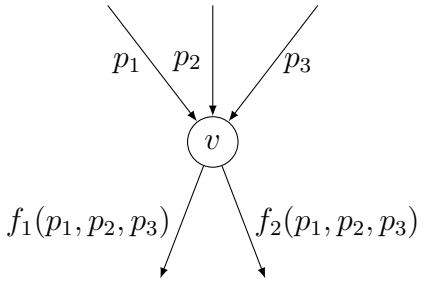


Figure 2.11: An example of encoding inside a vertex v. The data carried by the outgoing edges is determined by different functions of p_1, p_2 and p_3.

the data. To encode, the data on each outgoing edge of a vertex v is a function of the data on the incoming edges to v. An example is illustrated in Figure 2.11. Each receiver attempts to obtain the data sent from s by *decoding*. Ahlswede et al. [1] proved that using network coding, the multicasting rate is the smallest s-t edge connectivity among all receivers $t \in T$. In other words, it is given by $\min_{t \in T}\{\lambda_{s,t}\}$.

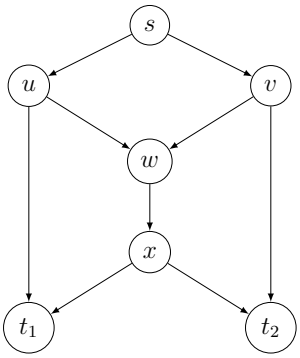


Figure 2.12: An example graph to show the advantage of network coding over routing.

Network coding can achieve higher multicasting rate than routing. Consider the graph shown in Figure 2.12. Using routing the multicasting rate is one. It is hard to obtain multicasting rate two because the edge (w, x) cannot transmit different data at the same time. We show how we can obtain a better multicasting rate using network coding. Figure 2.13 shows that the s-t_1 edge connectivity and the s-t_2 edge connectivity are both two. We present how this can be achieved in Figure 2.14. Here we let $a \oplus b$ to be $a + b$ modulo two. In the figure encoding is done in vertex w such that it adds the incoming data modulo two. It can be seen that t_1 receives data a and $a \oplus b$. Then t_1 can retrieve b by performing addition modulo two on the data, that is, $a \oplus (a \oplus b) = b$. Similarly t_2

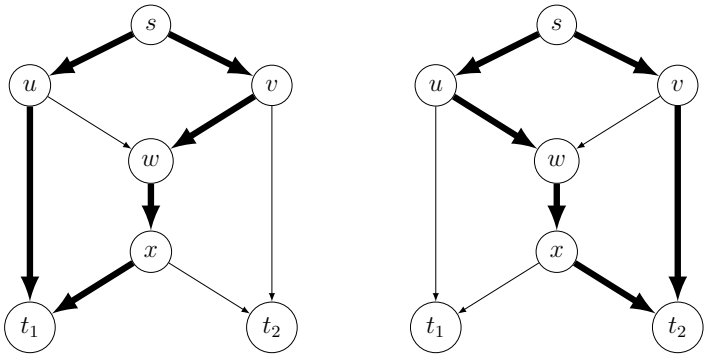


Figure 2.13: Consider the graph in Figure 2.12. There are two edge disjoint paths from s to t_1 and to t_2 respectively. The edge disjoint paths are respectively marked by thick edges in Figure 2.13a and Figure 2.13b. Therefore both edge connectivities are two.

can retrieve a and b, hence the multicasting rate in this network is two.

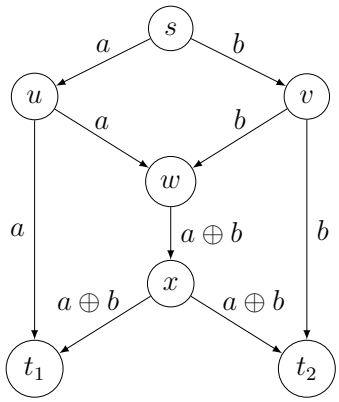


Figure 2.14: Using network coding in the graph presented in Figure 2.12, vertex w encodes the incoming data by addition modulo two (denoted as $\oplus$) and then transmits to the edge (w, x). Other vertices just forward their received data to their outgoing edges.

2.2.2 Linear Network Coding

In this section, we will introduce the concept of linear network coding, and see how to obtain optimal multicasting rate using linear network coding. Then we will present the concept of local encoding coefficients, global encoding vectors, linear network codes, and network coding solutions. At the end of this section we will review previous work about constructions of linear network code.

In Section 2.2.1, we have seen an example on how the optimal multicasting rate can be

achieved by network coding. Ahlswede et al. [1] only proved the existence of optimality but no constructive algorithm was given in their work. Later Li, Yeung and Cai [50] show that linear coding suffices to achieve the optimal multicasting rate for the multicasting problem.

Theorem 2.8 ([50], Theorem 3.3)**.** *Let $G = (V, E)$ be a directed acyclic graph, s be a source vertex, and $T \subset V$ be a set of receivers such that $s \notin T$. Let h be the optimal multicasting rate using network coding. Consider a multicasting transmission scheme over a large enough finite field $\mathbb{F}_q$, in which the data on an outgoing edge of a vertex is a linear combination of the data from the incoming edges to the same vertex. Then there exists such a scheme so that the multicasting rate is h.*

Theorem 2.8 says that the optimal multicasting rate can be achieved by linear coding in any directed graph. Figure 2.15 shows an example of linear encoding in a vertex.

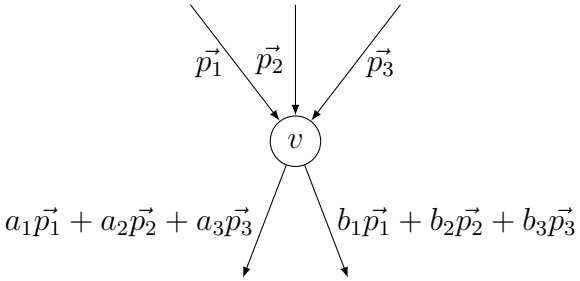


Figure 2.15: An example of linear encoding inside a vertex v. The data is represented by vectors. The data carried by the outgoing edges are determined by different linear combinations of p_1, p_2 and p_3, where $a_i, b_i \in \mathbb{F}_q$ for some finite field of size q.

Without loss of generality, we can assume that $d^{out}(s) = d^{in}(t) = h$ for every $t \in T$. In the actual transmission, the data is a $(h + 1)$-tuple. The first h values indicate the coefficients of the data sent from the source. The last value is the numerical value of the linear combination inside the data. Figure 2.16 shows the actual transmission for illustration. We omit the last value in the data for simplicity.

To decode in t_1, it has to solve the following system of linear equations:

$$\begin{cases} 1 \cdot a + 0 \cdot b = \alpha_{u,t_1} & \text{(Data inside edge } (u, t_1)) \\ 1 \cdot a + 1 \cdot b = \alpha_{x,t_1} & \text{(Data inside edge } (x, t_1)) \end{cases}$$

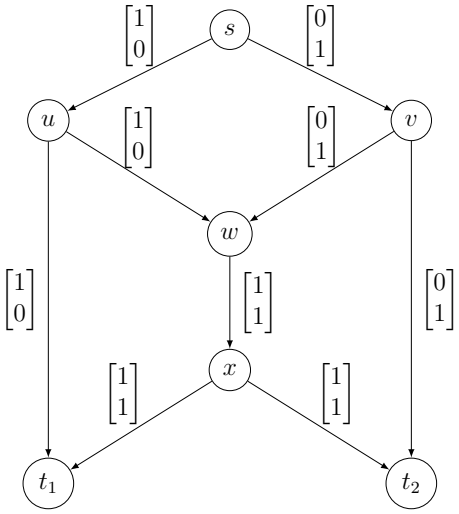


Figure 2.16: The global encoding vectors shown in the transmission.

where α_{u,t_1} and α_{x,t_1} are the last values contained in the corresponding data. Solving the equations yields $a = \alpha_{u,t_1}$ and $b = \alpha_{x,t_1} - \alpha_{u,t_1}$ respectively. Similarly t_2 can also obtain a and b by solving another set of linear equations. In general, the receiver needs to solve a system of linear equations of the form $Ax = b$, where x represents h units of data that need to be decoded. If A is of full rank, then the system has a unique solution, and the decoding is considered successful.

We will define the technical terms in the encoding of linear network coding. We define the data f_e transmitted inside an edge e as *global encoding vector*. For example, in Figure 2.16, $f_e = [1 \quad 0]^T$ for $e = (u, w)$ while $f_e = [1 \quad 1]^T$ for $e = (w, x)$. Encoding in each vertex can be characterized by a single matrix multiplication. Consider again the Figure 2.15. Suppose v stores a matrix

$$\begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{pmatrix},$$

then the global encoding vectors of the outgoing edges of v can be computed by

$$\begin{pmatrix} | & | & | \\ \vec{p_1} & \vec{p_2} & \vec{p_3} \\ | & | & | \end{pmatrix} \begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{pmatrix} = \begin{pmatrix} | & | \\ a_1\vec{p_1} + a_2\vec{p_2} + a_3\vec{p_3} & b_1\vec{p_1} + b_2\vec{p_2} + b_3\vec{p_3} \\ | & | \end{pmatrix},$$

and each column corresponds to the global encoding vector. We can do the same for all vertices: for every vertex v with indegree $d^{in}(v)$ and outdegree $d^{out}(v)$, there is a $d^{in}(v) \times d^{out}(v)$ matrix which characterizes the linear encoding process. We define each entry in the matrix as *local encoding coefficient*. Figure 2.17 shows the matrices of local encoding coefficients of the vertices of the graph shown in Figure 2.12. Note that for convenience, we can imagine that there are h distinct units of incoming data to the source vertex s so that its matrix of local encoding coefficients is well defined.

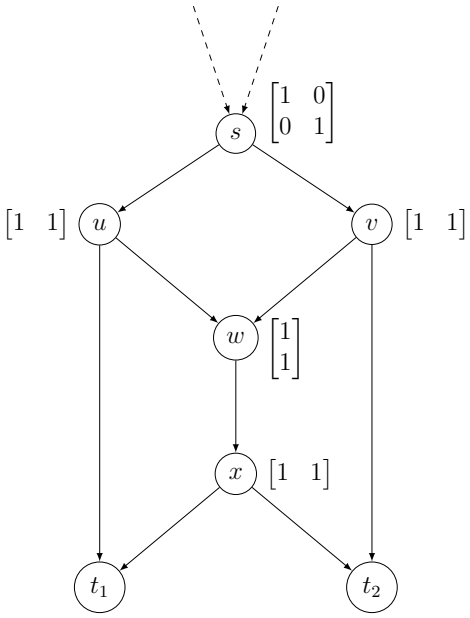


Figure 2.17: In this figure, the matrix of local encoding coefficients are indicated near the vertices. Follow the description of linear encoding by matrix multiplication, and assume that the dashed edges contain distinct data to s, all global encoding vectors can be deduced as in Figure 2.16.

A *linear network code* is defined by assigning values in $\mathbb{F}_q$ to all local encoding coefficients. Any linear network code is a (linear) *network coding solution* if the linear network code is such that every receiver can decode the h units of data successfully. That is the matrix of incoming vectors to any receiver is of full rank.

There are many results on the construction of network coding solution using linear network coding. Li et al. [50] proposed the first algorithm to construct a network coding solution, however the running time of their proposed algorithm is exponential. Later several polynomial time algorithms in constructing a network coding solution are proposed [39, 33, 46]. Jaggi and Sander et al. [39] proposed the first polynomial time

algorithm, which runs in $O(mkh+nk^2h^2(h+k))$ time, where k is the number of receivers and h is the optimal multicasting rate using network coding. The best known result is by Langberg et al. [46], who showed that a network coding solution can be constructed in $O(mkh + nk^2h^2 + h^4k^3(k + h))$ time. The improvement is made by reducing the number of intermediate vertices that perform encoding and reducing the time complexity of constructing a solution. Koetter and Médard [45] alternatively give an algebraic framework for linear network coding, which is relatively simpler.

2.2.3 Random Linear Network Coding

In the random network coding algorithm, the local encoding coefficients are random values in a large enough field. It is shown [34, 35] that if the optimal multicasting rate is h, then with high probability the random network coding algorithm can achieve such rate. The main advantage of the random network coding algorithm is that assigning local encoding coefficients can be done in a decentralized manner, and this is very useful in practical applications.

We are going to prove the optimality of multicasting rate using the random network coding algorithm. The proof is based on [35].

Theorem 2.9 ([35], Theorem 3)**.** *Given a directed acyclic graph $G = (V, E)$, a source vertex s, and a set $T \subset V$ of k receivers. Also let h be the optimal multicasting rate using network coding. Let A_t be a $h \times d^{in}(t)$ matrix for every $t \in T$ such that each column vector f_e corresponds to the data on edge $e \in \delta^{in}(t)$. If all local encoding coefficients are drawn randomly from a large enough finite field $\mathbb{F}_q$, and each pair of coefficients are independent, then with high probability A_t is of rank h for all $t \in T$.*

Proof. Now let us treat the local encoding coefficients in each vertex as variables. Let $l(v)$ be the label of vertex v such that

$$l(v) = \begin{cases} 1 & \text{if } v = s \\ \max_{uv \in \delta^{in}(v)} l(u) + 1 & \text{otherwise} \end{cases}.$$

Then by inductive argument it can be easily seen that each entry of the global encoding

vector of $e \in \delta^{out}(v)$ is a multivariate polynomial of local encoding coefficients with total degree at most $l(v)$, since encoding involves addition of multiple global encoding vectors with each entry being multiplied by at most one local encoding coefficient, hence the label of a vertex implicity characterizes the total degree.

As $l(v) \leq n$ for all $v \in V$, each entry in A_t is a multivariate polynomial of local encoding coefficients with degree at most n. Therefore $\det(A_t)$ is just a multivariate polynomial of local encoding coefficients of degree at most hn since A_t is of size $h \times h$. Assume that $\det(A_t) \not\equiv 0$, then we can apply Schwartz-Zippel Lemma such that if $q \leq hn^3$ and we assign random values to the local encoding coefficients independently from $\mathbb{F}_q$, then $\det(A_t) = 0$ with probability at most $\frac{1}{n^2}$. Hence every receiver t has $\det(A_t) \neq 0$ with probability at least $1 - \frac{1}{n}$ by union bound.

It remains to show that $\det(A_t) \not\equiv 0$ for every receiver t. It suffices to give one assignment to the local encoding coefficients such that $\det(A_t) \neq 0$. Since by Theorem 2.8 there are h edge disjoint paths from s to t, for each pair of incoming edge e and outgoing edge e' such that they belong to the same edge disjoint path, we assign the corresponding local encoding coefficient to be one, otherwise assign zero. Then one can easily verify that eventually t receives h distinct unit vectors, hence $A_t = I_h$ where I_h is a $h \times h$ identity matrix. Therefore $\det(A_t) = 1$ and the proof is completed. □

2.3 Algebraic Tools

In this section, we will present various definitions and theorems on linear algebra. We will first describe some matrix notations and some basic algorithms. Then we will introduce nested dissection, which is a method of solving system of linear equations.

2.3.1 Linear Algebraic Algorithms

Throughout the thesis, we would use the following notations for any matrix M. If M is a matrix, then we can describe any submatrix of M as $M_{S,T}$, where S and T are the row and column subsets of M. If we want to include all rows (or columns) we shall write it as $M_{*,T}$ (or $M_{S,*}$). So $M_{*,*}$ is identical to M. Let $\vec{e}_i$ be the i-th vector in the standard

basis.

2.3.1.1 Matrix Multiplications, Determinant and Inverse

Given two $n \times n$ matrices with entries in a field $\mathbb{F}$ of size $poly(n)$, the matrix multiplication operation can be done in $O(n^\omega)$ time [16] where $\omega < 2.38$. For an $n \times n$ matrix, it is known that the operations of computing the determinant, computing the rank and computing the inverse can all be done in the same time bound as one matrix multiplication [11, 31]. Given an $n \times m$ matrix A and an $m \times n$ matrix B, we can use fast matrix multiplication to improve the running time from the naïve $O(n^2 m)$ algorithm. The procedure is to break A and B into $k = \lfloor \frac{m}{n} \rfloor$ submatrices, each of them is of size $n \times n$, so that:

$$AB = \begin{pmatrix} A_1 & A_2 & \cdots & A_k \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_k \end{pmatrix} = \sum_{i=1}^{k} A_i B_i.$$

Now $A_i B_i$ can be computed in $O(n^\omega)$ time and thus AB can be computed in $O(kn^\omega) = O(\frac{m}{n} n^\omega) = O(mn^{\omega-1})$ time.

2.3.1.2 Schwartz-Zippel Lemma and Matrix of Indeterminates

Schwartz-Zippel Lemma provides us a randomized algorithm to test whether a polynomial is identically equal to zero or not, with high probability.

Theorem 2.10 ([57] Theorem 3.2 (Schwartz-Zippel Lemma))**.** *Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a non-zero polynomial of degree $d \geq 0$ over a field $\mathbb{F}$. Let S be a finite subset of $\mathbb{F}$ and let $r_1, r_2, \ldots, r_n$ be selected randomly from S. Then*

$$\Pr[P(r_1, r_2, \ldots, r_n) = 0] \leq \frac{d}{|S|}.$$

Proof. The proof is by mathematical induction on the number of variables. The base case is $n = 1$. Here $P(x)$ is a degree d polynomial with at most d roots, hence $Pr[P(r) = 0] \leq \frac{d}{|S|}$.

Now assume the theorem holds for all polynomials with $n-1$ variables where $n > 1$. We can write P as

$$P(x_1, ..., x_n) = \sum_{i=0}^{d} x_1^i P_i(x_2, ..., x_n).$$

If $P \not\equiv 0$ then there exists i such that $P_i(x_2, ..., x_n) \not\equiv 0$. Consider the largest such i. Randomly pick $r_1, ..., r_n \in S$. By the induction hypothesis, $Pr[P_i(r_1, r_2, ..., r_n) = 0] \leq \frac{d-i}{|S|}$ since P_i has degree $d-i$. If $P_i(r_1, r_2, ..., r_n) \neq 0$, then $P(x_1, r_2, ..., r_n)$ is now of degree i with one variable. Hence $Pr[P(r_1, r_2, ..., r_n) = 0 | P_i(r_2, ..., r_n) \neq 0] \leq \frac{i}{|S|}$. Finally,

$$\begin{aligned}
& Pr[P(r_1, r_2, ..., r_n) = 0] \\
= \; & Pr[P(r_1, r_2, ..., r_n) = 0 | P_i(r_2, ..., r_n) = 0] + Pr[P(r_1, r_2, ..., r_n) = 0 | P_i(r_2, ..., r_n) \neq 0] \\
\leq \; & Pr[P_i(r_2, ..., r_n) = 0] + Pr[P(r_1, r_2, ..., r_n) = 0 | P_i(r_2, ..., r_n) \neq 0] \\
\leq \; & \frac{d-i}{|S|} + \frac{i}{|S|} = \frac{d}{|S|}
\end{aligned}$$

□

Let $\mathbb{F}$ be a field, and let $\mathbb{F}(x_1, \ldots, x_m)$ be the field of rational function over $\mathbb{F}$ with indeterminates $\{x_1, x_2, \ldots, x_m\}$. A matrix with entries in $\mathbb{F}(x_1, \ldots, x_m)$ is called a matrix of indeterminates. A matrix M of indeterminates is non-singular if and only if its determinant is not the zero function. To check if an $n \times n$ matrix M with indeterminates is non-singular, one can substitute each x_i with a random value in $\mathbb{F}_q$ and call the resulting matrix M'. By the Schwartz-Zippel Lemma, if M is non-singular then $\det(M')$ is zero with probability at most n/q. Hence, by setting $q = n^c$ for a large constant c, this gives a randomized algorithm with running time $O(n^\omega)$ to test if M is non-singular with high probability.

2.3.2 Nested Dissection

Nested dissection is a method in solving certain kinds of system of linear equations. In this section we will first review Gaussian elimination, which is a classical technique of solving linear equations. Then we will define positive definite matrix, which is required to apply the nested dissection method. After that we will introduce the notion of elimination

ordering of a matrix. Then we will describe generalized nested dissection and we will discuss its applications to various classes of undirected graphs. We end this section by briefly stating the limitations of generalized nested dissection and presenting its recent extensions.

In linear algebra, solving a system of n linear equations with n unknowns can be represented in matrix form $Ax = b$, where A is an $n \times n$ matrix, x is a solution vector of n unknowns. We assume all entries in A, b and x are real. The solution x is unique if and only if A is of full rank. Then to obtain the value of entries in x, one classical method is to perform Gaussian elimination on A. After performing elimination on A, every entry $A_{i,j}$ where $i > j$ must be zero, as shown below:

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ 0 & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{n,n} \end{pmatrix}.$$

Ordinary Gaussian elimination algorithm takes $O(n^3)$ time. A variation of Gaussian elimination that also takes $O(n^3)$ time is known as Gaussian-Jordan elimination, which eliminates until A becomes a diagonal matrix, as shown below:

$$A = \begin{pmatrix} A_{1,1} & 0 & \cdots & 0 \\ 0 & A_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{n,n} \end{pmatrix}.$$

We now define positive definite matrix, and discuss how we can improve the running time of elimination algorithm in this class of matrices. We assume that the elimination algorithm we use is Gaussian-Jordan.

Definition. *An matrix $A \in \mathbb{R}^{n \times n}$ is called positive definite if for every $x \in \mathbb{R}^n$ such that $x \neq \vec{0}$ we have $x^T Ax > 0$.*

Positive definite matrix guarantees the existence and uniqueness of the solution in a system of linear equations.

Theorem 2.11. *If a matrix $A \in \mathbb{R}^{n\times n}$ is positive definite, then* $\det(A) \neq 0$.

Proof. Assume $\det(A) = 0$. Then it is equivalent to $Ax = \vec{0}$ for some $x \in \mathbb{R}^n$, and $x \neq \vec{0}$. It implies that

$$x^T Ax = x^t(\vec{0}) = 0.$$

However it means that A is not positive definite, thus a contradiction. □

Corollary 2.12. *If a matrix $A \in \mathbb{R}^{n\times n}$ is positive definite, then the system of linear equations $Ax = b$ must have exactly one solution.*

We will give some graph theoretic interpretations of Gaussian elimination. We assume that no numerical cancellation occurs during elimination. We associate any positive definitie matrix A with its *underlying graph* $G_A = (V, E)$, which is an undirected graph so that there are n vertices, and $e = (u, v) \in E$ if both $A_{v,u} \neq 0$ and $A_{u,v} \neq 0$. Note that we ignore the self loops presented in $A_{i,i}$. In addition, A must be symmetric, meaning that $A_{i,j} = A_{j,i}$ for every i and j. When performing elimination at $A_{i,i}$, it is equivalent to removing vertex i in G_A since after elimination, $A_{i,j} = A_{j,i} = 0$ for $j \neq i$, which implies that vertex i does not have edges connecting to any other vertices. Fill-in occurs at $A_{j,k}$ and $A_{k,j}$ when $A_{i,j} \neq 0$ and $A_{i,k} \neq 0$. We see that it is equivalent to add an edge (j, k) before vertex i is removed, if (i, j), $(i, k) \in E$. Then it can be seen that when eliminating vertex i in G_A, there will be $\binom{d(i)}{2}$ fill-ins. See Figure 2.18 for a simple example.

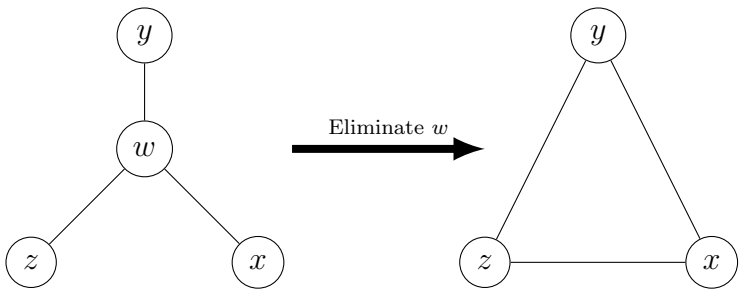


Figure 2.18: An example of showing elimination in a graph. When vertex w is eliminated in the left graph, the edges are added between any pair of vertices of x, y and z. The result is shown on the right graph.

Definition. *An elimination ordering is a bijection $\tau : \{1, 2, \cdots, n\} \to V$ so that the elimination of the entries of A is done in the order $A_{\tau(1),\tau(1)}, \cdots A_{\tau(n),\tau(n)}$. The underlying*

graph G_A after the k-th elimination according to τ is denoted as $G^k_{A,\tau} = (V, E_k)$, so that $G^0_{A,\tau} = G_A$ and $G^n_{A,\tau}$ has no edges. E_k is the set of edges after performing upto k-th elimination according to the ordering defined in τ.

Different elimination orderings produce different number of fill-ins. Consider again the graph presented in Figure 2.18. An elimination ordering $\{w, x, y, z\}$ would produce three fill-ins, which is indicated in Figure 2.18. However, if we consider any elimination ordering such that vertex w is the last eliminated vertex, then there are no fill-ins at all. For example, consider the elimination ordering $\{z, y, x, w\}$.

Let $d_k(i)$ be the degree of vertex i in the graph $G^k_{A,\tau}$. Then one can verify that a particular elimination ordering τ would lead to the total number of fill-ins F_τ such that

$$F_\tau = \sum_{i=1}^{n} \binom{d_i(\tau(i))}{2}.$$

Ideally we wish to find τ such that F_τ is minimized, however it is shown to be NP-complete [24].

Nested dissection is a technique to reduce the number of fill-ins if the underlying graph of a matrix has good separators. Recall from Section 2.1.7, let Z be a separator of G_A such that its removal in G_A results in two separated subgraphs with vertex sets X and Y. The main idea of nested dissection is that eliminating vertices in Z first would produce more fill-ins than eliminating them at last. We again consider the graph in Figure 2.18, we see that $Z = \{w\}$ is a good separator. Hence, as explained before, that eliminating it last gives less fill-ins. The study of nested dissection is pioneered by George [25].

Generalized nested dissection is the nested dissection method in the graph with good separators. As discussed in Section 2.1.7, if G_A is planar, we can recursively decompose G_A using $(O(\sqrt{n}), 2/3)$-separators according to Theorem 2.4. As such we obtained a weak-separator tree T_A and we perform elimination of A in the bottom up fashion according to T_A. Lipton, Rose and Tarjan [51] first proposed this elimination algorithm in a hereditary class of graphs that has $(O(\sqrt{n}), \alpha)$-separation where $\alpha \in (0, 1)$. They proved the fill-in bound and the running time of the algorithm as below.

Theorem 2.13 (Generalized Nested Dissection, [51])**.** *Let $A \in \mathbb{R}^{n\times n}$ be a positive definite*

matrix such that the underlying undirected graph G_A is in a hereditary class of graphs that has a $(O(\sqrt{n}), \alpha)$-separation and can be found in $O(n^\gamma)$ time. Then performing elimination in A results in $O(n \log n)$ fill-ins and runs in $O(n^{\omega/2} + n^\gamma + n \log n)$ time.

There is another simpler algorithm of generalized nested dissection, which is given by Gilbert and Tarjan [27]. Note that George's work [25] is a special case of generalized nested dissection.

There are certain hereditary classes of graphs that can apply Theorem 2.13. As mentioned in Section 2.1.7, the class of planar graphs, the class of bounded genus graphs, and the class of fixed minor free graphs have a $(O(\sqrt{n}), \alpha)$-separation (See Section 2.1.1 for their definitions). In both planar graph and bounded genus graph the separators can be constructed in linear time [52, 26], therefore nested dissection runs in $O(n^{\omega/2})$ time for both cases. For fixed minor free graph, there is an $O(n^{1.5})$ algorithm [4] to find the separator and the nested dissection runs in $O(n^{1.5})$ time.

There are several restrictions when applying nested dissection algorithms. They are only applicable to a matrix A which satisfies the following properties:

1. A is symmetric, positive definite.

2. Every entry in A is in $\mathbb{R}$.

In the application of this thesis, we are dealing with matrices that are neither positive definite, symmetric nor every entry is real. Interestingly, Alon and Yuster [5] extended the nested dissection method such that it works for any matrix (not necessarily symmetric) and for any field. It only requires that the underlying graph (now an edge (i, j) is presented if either $A_{i,j} \neq 0$ or $A_{j,i} \neq 0$) has a $(O(n^\beta), \alpha)$-weak separator tree. In the following a family of graphs is δ-sparse if any n-vertex graph in this family has at most δn edges.

Theorem 2.14 (Alon, Yuster [5])**.** *Let $\mathcal{F}$ be a δ-sparse family of graphs with an $O(n^\gamma)$ time algorithm to find an $(O(n^\beta), \alpha)$-weak separator tree where α is a positive constant smaller than 1. Given a system of linear equations $Ax = b$ where $A \in \mathbb{F}^{n \times n}$ is non-singular, $b \in \mathbb{F}^n$ and the underlying graph of A is in $\mathcal{F}$, there is a randomized algorithm that finds the unique solution of the system in $O(n^{\omega\beta} + n^\gamma + n \log n)$ time.*

One consequence is that one can apply the nested dissection method to the directed version of the previously mentioned classes of graphs, and we will use this in Section 3.4 to obtain faster algorithms for computing edge connectivities.

Chapter 3

Algorithms for Graph Connectivities

The results of this chapter are based on joint work with Ho Yee Cheung and Lap Chi Lau [15].

3.1 Introduction

Graph connectivity is a basic concept that measures the reliability and efficiency of a graph. The edge connectivity from vertex s to vertex t is defined as the size of a minimum s-t cut, or equivalently the maximum number of edge disjoint paths from s to t. Computing edge connectivities is a classical and well-studied problem in combinatorial optimization. Most known algorithms to solve this problem are based on network flow techniques (see e.g. [58]).

The fastest algorithm to compute s-t edge connectivity in a simple directed graph by Even and Tarjan runs in $O(\min\{m^{1/2}, n^{2/3}\} \cdot m)$ time, where m is the number of edges and n is the number of vertices. To compute the edge connectivities for many pairs, however, it is not known how to do it faster than computing edge connectivity for each pair separately, even when the pairs share the source or the sink. For instance, it is not known how to compute all pairs edge connectivities faster than computing s-t edge connectivity for $\Omega(n^2)$ pairs. This is in contrast to the problem in undirected graphs, where all pairs edge connectivities can be computed in $\tilde{O}(mn)$ time by constructing a Gomory-Hu tree [8].

Network coding is an innovative method to transmit information in a computer network. The fundamental result is a max-information-flow min-cut theorem for multicasting [1]: if the edge connectivity between the source vertex s to each sink vertex t_i is at least k, then one can transmit k units of information to all sink vertices simultaneously, by performing encoding and decoding at the vertices. An elegant algebraic framework has been developed to construct efficient network coding schemes for multicasting [50, 45].

In this thesis, we use the techniques developed in network coding to obtain a new algebraic formulation for computing edge connectivities. This reduces the problem of computing edge connectivities to solving systems of linear equations, and opens up new directions to design algorithms for the problem. One advantage is that the edge connectivities from a source vertex to all other vertices can be computed simultaneously (as in the max-information-flow min-cut theorem). This leads to faster algorithms for computing single source edge connectivities and all pairs edge connectivities, and in some settings the amortized time to compute the edge connectivity for one pair is sublinear. In the process we have also found an interesting use of expanders and superconcentrators to design fast algorithms for graph connectivity problems.

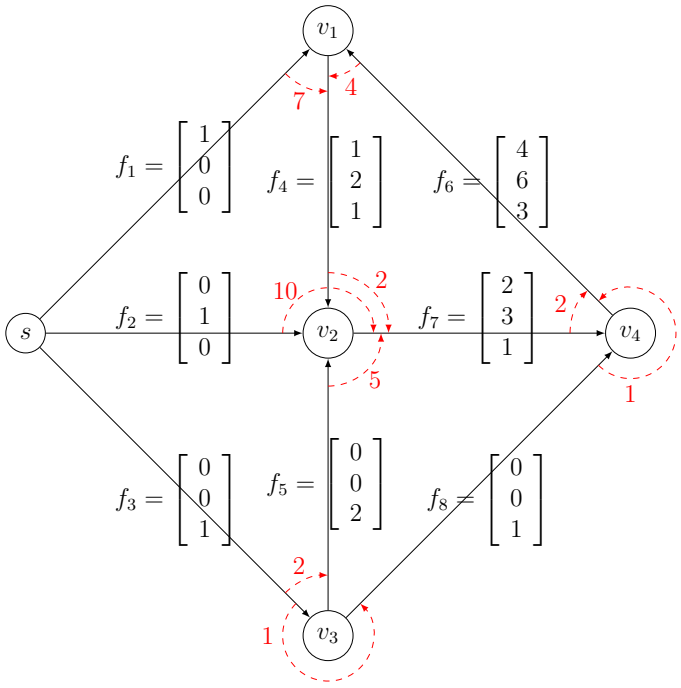
3.1.1 Our Results

Our new algebraic formulation for computing edge connectivities is inspired by the random linear coding algorithm [35] in constructing network codes. Let $G = (V, E)$ be a directed graph. Let s be the source vertex with out-degree d, with outgoing edges $e_1, \ldots, e_d$. For each edge $e \in E$ we associate a vector f_e of dimension d, where each entry in f_e is an element from a large enough finite field $\mathbb{F}$. The vectors f_e are required to satisfy the following properties: (1) the vectors on $e_1, \ldots, e_d$ are linearly independent, and (2) the vector on an edge $e = (v, w)$ is a random linear combination of the incoming vectors of v. Once we obtain the vectors, we can compute the edge connectivities from the source vertex as follows.

Theorem 3.1 (Informal statement)**.** *With high probability there is a unique solution to the vectors, and the edge connectivity from s to t is equal to the rank of the incoming*

vectors of t for any $t \in V - s$.

See Figure 3.1 for an example and Theorem 3.5 for the formal statement. This formulation is previously known for directed acyclic graphs only [38, 60]. For general directed graphs, network coding schemes require convolution codes [18, 49], and it is not known how to use these to compute edge connectivities. Our contribution is to come up with this simple formulation that can be used to compute edge connectivities only.



$$f_4 = 7 \cdot f_1 + 4 \cdot f_6$$
$$f_5 = 2 \cdot f_3$$
$$f_6 = 2 \cdot f_7 + 1 \cdot f_8$$
$$f_7 = 2 \cdot f_4 + 10 \cdot f_2 + 5 \cdot f_5$$
$$f_8 = 1 \cdot f_3$$

Figure 3.1: In this example three independent vectors f_1, f_2 and f_3 are sent from the source s. Other vectors are a linear combination of the incoming vectors, according to the random coefficients on the dotted lines. All operations are done in the field of size 11. To compute the edge connectivity from s to v_2, for instance, we compute the rank of (f_2, f_4, f_5) which is 3 in this example.

The algebraic formulation reduces the problem of computing edge connectivities to solving systems of linear equations. We call the step to compute the vectors f_e as the *encoding* step, and the step to compute the ranks as the *decoding* step. The formulation has the advantage that after the encoding step has been done, the edge connectivities from the source vertex s to all other vertices can be computed readily.

We first present algorithmic results on single source edge connectivities and then algorithmic results on all pairs edge connectivities.

In directed acyclic graphs, the encoding step can be implemented directly without solving linear equations, but the resulting algorithm is not competitive to the existing algorithms. By a simple transformation using superconcentrators [61, 36], we show how to implement the encoding step optimally by this direct approach. This can be used to improve the random linear coding algorithm [35] for network coding. Also we obtain a faster algorithm for computing single source edge connectivities in directed acyclic graphs. The algorithm runs in linear time when d is a constant, and by a simple reduction this can be used to return all vertices with edge connectivity at most d from the source s in linear time.

Theorem 3.2. *Given a simple directed acyclic graph and a source vertex s with outdegree d, the encoding step can be done in $O(dm)$ time, and the edge connectivities from s to all vertices can be computed in $O(d^{\omega-1}m)$ time, where $\omega \approx 2.38$ is the matrix multiplication exponent.*

In some graphs the system of linear equations can be solved more efficiently. For instance, we can use a recent result of Alon and Yuster [5] to perform the encoding step faster in directed planar graphs with constant maximum degree. The best known algorithm to compute single source edge connectivities in directed planar graphs requires $O(n^2)$ time, by using a $O(n)$ time algorithm to compute s-t edge connectivity [10].

Theorem 3.3. *Given a simple directed planar graph with constant maximum degree and a source vertex s, the edge connectivity from s to all vertices can be computed in $O(n^{\omega/2})$ time.*

For general directed graphs, we show that all pairs edge connectivities can be computed in one matrix inverse time, instead of solving the linear equations for each source vertex separately. The algorithm is faster when the graph has $O(n^{1.93})$ edges, for example when $m = O(n)$ it takes $O(n^\omega)$ time while the best known algorithm takes $O(n^{3.5})$ time.

Theorem 3.4. *Given a simple directed graph, the edge connectivities between all pairs of vertices can be computed in $O(m^\omega)$ time where m is the number of edges in the graph.*

The idea of transforming the graph using superconcentrators can also be used in other graph connectivity problems. In Section 3.6 we show how to use expanders and superconcentrators to speedup the algorithms for finding edge splitting-off operations preserving edge connectivities in undirected and directed graphs.

3.1.2 Related Work

The standard way to solve the s-t edge connectivity problem in directed graphs is by network flow techniques. For simple directed graphs, the best known algorithm is a $O(\min\{n^{2/3}, m^{1/2}\} \cdot m)$ time algorithm [19] by the blocking flow method. As mentioned previously, in directed graphs, it is not known how to compute all pairs edge connectivities faster than computing s-t edge connectivity for $\Omega(n^2)$ pairs separately. This is in contrast to the problem in undirected graphs, where all pairs edge connectivities can be computed in $\tilde{O}(mn)$ time by constructing a Gomory-Hu tree [8], much faster than computing edge connectivities for $\Omega(n^2)$ pairs.

There are many improvements in special cases of the s-t edge connectivity problem in directed graphs. For bipartite matching, the best known algorithm is a $O(m\sqrt{n})$ time algorithm [37] by the blocking flow method, and a $O(n^\omega)$ time algorithm [55, 32] by algebraic techniques. It is known that the bipartite matching problem is equivalent to the s-t vertex connectivity problem in directed graphs, and so the above results hold for the latter problem as well [14]. For simple undirected graphs, there is a $O(n^{3/2}\sqrt{m})$ time algorithm [29] by a combination of the blocking flow method and a graph sparsification technique, and the best known algorithm is a $\tilde{O}(n^{2.2})$ time algorithm [43] by extending [29] with random sampling. In directed planar graphs, there is an optimal $O(n)$ time algorithm for computing s-t edge connectivity [10].

For network coding, the max-information-flow min-cut theorem for multicasting is first proved by an information theoretical argument [1]. Later on it is shown that linear network coding is enough to achieve the max-flow min-cut theorem for multicasting [50] and an algebraic framework is developed [45]. Then a polynomial time deterministic algorithm is obtained to construct optimal linear coding schemes for multicasting in directed acyclic graphs [39], and in general directed graphs using

convolution codes [18, 49]. Subsequently a simple polynomial time randomized algorithm is obtained for constructing optimal linear coding schemes for multicasting [35], and our algorithm is based on this approach.

3.1.3 Techniques

The starting observation is that the random linear coding algorithm [35] for network coding does not require the knowledge of the graph topology, and it could be used to compute the edge connectivities from the source to the sinks. Actually this observation was already made for directed acyclic graphs in earlier work [38, 60]. For general directed graphs, however, network coding schemes are more complicated (even for random linear coding) as convolution codes are required [18, 49], and it is not known how to use these to compute edge connectivities. Our contribution is to come up with a simple formulation that can be used to compute edge connectivities only. The simple coding scheme (without using convolution codes) also allows us to design more efficient algorithms. The proof extends the ideas developed in the random linear coding algorithm.

We show a simple transformation using expanders and superconcentrators [61, 36] to design fast algorithms for some graph connectivity problems. In Eulerian directed graphs, the idea is to replace a vertex of indegree d and outdegree d by a superconcentrator with d inputs and d outputs. This reduces the maximum degree of the graph significantly, while preserving the edge connectivities and only increasing the number of vertices moderately. For random linear coding, using the direct algorithm in the resulting graph gives an optimal algorithm in the original graph. For edge splitting-off, using the straightfoward algorithm in the resulting graph gives a considerable improvement upon the same algorithm in the original graph. In undirected graphs, we can use constant degree expander graphs for the same purpose.

For all pairs edge connectivities, we observe that if we change the source vertex the system of linear equations is similar, and so we could compute the n single source edge connectivities in one matrix inversion time.

3.1.4 Organization

We present the algebraic formulation in Section 3.2 and prove a formal statement of Theorem 3.1. In Section 3.3 and 3.4 we present algorithms for single source edge connectivities and prove Theorem 3.2 and 3.3. In Section 3.5 we present algorithms for all pairs edge connectivities and prove Theorem 3.4. Finally we show the use of expanders and superconcentrators in the edge splitting-off problem in Section 3.6.

3.2 New Algebraic Characterization

We define the algebraic formulation for computing graph connectivities formally. Given a directed graph $G = (V, E)$ and a specified source vertex s, we are interested in computing the edge connectivities from s to other vertices. Let $m = |E|$ and $E = \{e_1, e_2, \ldots, e_m\}$. Let $d = d^{out}(s)$ and $\delta^{out}(s) = \{e_1, \ldots, e_d\}$. Let $\mathbb{F}$ be a finite field. For each edge $e \in E$, we associate a *global encoding vector* $f_e \in \mathbb{F}^d$ of dimension d where each entry is in $\mathbb{F}$. We say a pair of edges e' and e are *adjacent* if the head of e' is the same as the tail of e, i.e. $e' = (u, v)$ and $e = (v, w)$ for some $v \in V$. For each pair of adjacent edges e' and e, we associate a *local encoding coefficient* $k_{e',e} \in \mathbb{F}$. We assume that $k_{e',e} = 0$ for every pair of edges e' and e that are not adjacent. Given the local encoding coefficients for all pairs of adjacent edges in G, we say that the global encoding vectors are a *network coding solution* if the following two sets of equations are satisfied:

1. For each edge $e_i \in \delta^{out}(s)$, we have $f_{e_i} = \sum_{e' \in \delta^{in}(s)} k_{e',e_i} \cdot f_{e'} + \vec{e}_i$, where $\vec{e}_i$ is the i-th vector in the standard basis.

2. For each edge $e = (v, w)$ with $v \neq s$, we have $f_e = \sum_{e' \in \delta^{in}(v)} k_{e',e} \cdot f_{e'}$.

An example is depicted in Figure 3.1. Note that not every possible choices of the local encoding coefficients can result in a network coding solution. But we can guarantee the existence and the uniqueness of such solution by the following theorem, as the formal statement of Theorem 3.1.

Theorem 3.5. *Let $\mathbb{F}$ be a finite field with $|\mathbb{F}| = \Omega(m^{c+3})$ for some integer c. If we choose each local encoding coefficient independently and uniformly at random from $\mathbb{F}$, then with*

probability at least $1 - O(1/m^c)$*:*

1. *There is a unique network coding solution for the global encoding vectors* f_e*.*
2. *For* $t \in V - s$*, let* $\delta^{in}(t) = \{a_1, \ldots, a_l\}$*, the edge connectivity from* s *to* t *is equal to the rank of the matrix* $(f_{a_1}, f_{a_2}, \ldots, f_{a_l})$*.*

We will prove Theorem 3.5 in the remaining of this section. First we need to state a matrix form of a network coding solution.

Claim 3.6. *Let* F *be a* $d \times m$ *matrix such that*

$$F = \begin{pmatrix} | & & | \\ f_{e_1} & \cdots & f_{e_m} \\ | & & | \end{pmatrix}.$$

If all f_{e_i} *in* F *are a network coding solution, then we have* $F = FK + H_s$*, where* K *is an* $m \times m$ *matrix such that* $K_{i,j} = k_{e_i,e_j}$*. Also,* H_s *is a* $d \times m$ *matrix with exactly* d *columns that form a basis of* d*-dimensional space.*

Proof. Let

$$F' = FK = \begin{pmatrix} | & & | \\ f'_{e_1} & \cdots & f'_{e_m} \\ | & & | \end{pmatrix}.$$

If the vectors in F are a network coding solution, then for every edge $e \notin \delta^{out}(s)$, $f'_e = f_e$. For every edge $e = (v, w)$, we have

$$f'_e = \sum_{e' \in E} f_{e'} k_{e',e}\ .$$

Since $k_{e',e} = 0$ for all e' such that e' and e are not adjacent, we can rewrite f'_e as

$$f'_e = \sum_{e' \in \delta^{in}(v)} f_{e'} k_{e',e}\ .$$

By the second requirement of network coding solution, we can conclude that $f'_e = f_e$ for every $e \notin \delta^{out}(s)$.

For every $e_i \in \delta^{out}(s)$, $f_{e_i} - f'_{e_i} = \vec{e_i}$. Therefore if we add F' by an additional $d \times m$ matrix H_s such that the column which corresponds to the edge e_i is $\vec{e_i}$, while the other columns are zero vectors, then $F' + H_s$ also satisfies the first requirement of network coding solution. Therefore the vectors in $F' + H_s$ are a network coding solution. Since the vectors in F are also a network coding solution, we can conclude that

$$F = F' + H_s = FK + H_s. \tag{3.1}$$

□

Without loss of generality, we can assume that the first d columns of H_s are distinct vectors from the standard basis. For this to be true we can permute some columns in F and can permute columns and rows in K, such that we can assume that (3.1) is in the following form

$$\begin{pmatrix} & | & \\ \cdots & f_{e_j} & \cdots \\ & | & \end{pmatrix} = \begin{pmatrix} & | & \\ \cdots & f_{e_j} & \cdots \\ & | & \end{pmatrix} \begin{pmatrix} & k_{e_1,e_j} & \\ & \vdots & \\ \cdots & k_{e_i,e_j} & \cdots \\ & \vdots & \\ & k_{e_m,e_j} & \end{pmatrix} + \begin{pmatrix} & & & & & \\ | & & | & | & & | \\ \vec{e}_1 & \cdots & \vec{e}_d & \vec{0} & \cdots & \vec{0} \\ | & & | & | & & | \\ & & & & & \end{pmatrix}.$$

The equation (3.1) can be rewritten as $F(I - K) = H_s$. Hence to prove the first part of Theorem 3.5, we will prove in the following lemma that $(I - K)$ is non-singular with high probability. Then (3.1) can be formulated as $F = H_s(I - K)^{-1}$, which implies that the global encoding vectors are uniquely determined.

Lemma 3.7. *Given the conditions in Theorem 3.5, the matrix $(I - K)$ is non-singular with probability at least $1 - O(1/m^{c+2})$.*

Proof. Since the diagonal entries of K are zero, the diagonal entries of $I - K$ are all one. By treating each entry of K as an indeterminate $K_{i,j}$, it follows that $\det(I - K) = 1 + p(\cdots, K_{i,j}, \cdots)$ where $p(\cdots, K_{i,j}, \cdots)$ is a polynomial of the indeterminates with total degree at most m. Note that $\det(I - K)$ is not a zero polynomial since there is a constant term. Hence, by the Schwartz-Zippel Lemma, if $|\mathbb{F}| = \Omega(m^{c+3})$ and each $K_{i,j}$

is a random element in $\mathbb{F}$, then $\det(I - K) = 0$ with probability at most $O(1/m^{c+2})$, proving the lemma. □

After we obtained the global encoding vectors, we would like to show that the edge connectivities can be determined from the ranks of these vectors. Consider a vertex $t \in V - s$. Let $\delta^{in}(t) = \{a_1, \ldots, a_l\}$ and let M_t be the $d \times l$ matrix $(f_{a_1}, f_{a_2}, \ldots, f_{a_l})$. Let the edge connectivity from s to t be $\lambda_{s,t}$. We prove in the following lemma that $\text{rank}(M_t) = \lambda_{s,t}$ with high probability.

Lemma 3.8. *Given the conditions of Theorem 3.5, we have* $\text{rank}(M_t) = \lambda_{s,t}$ *with probability at least* $1 - O(1/m^{c+1})$.

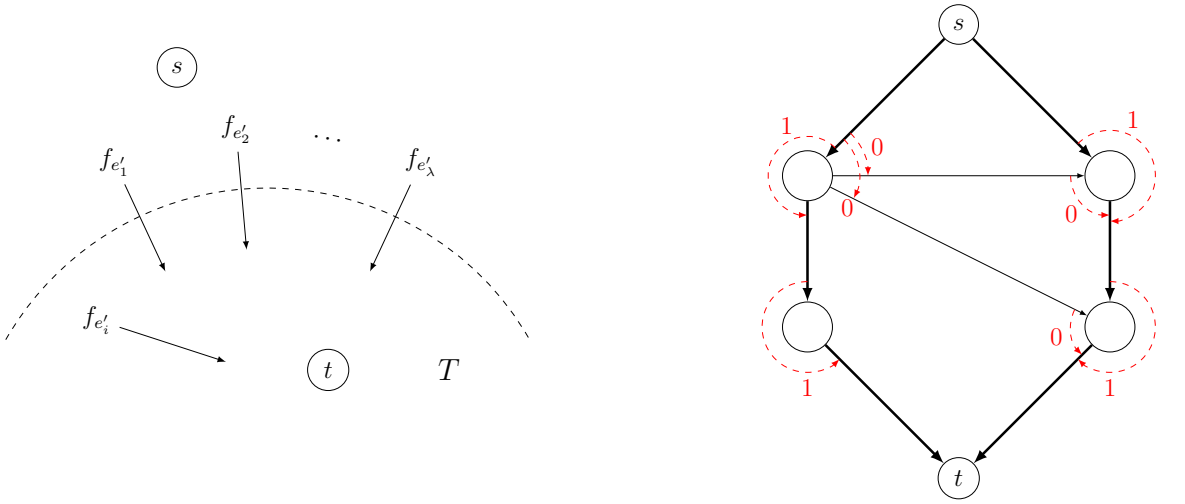
Proof. We will first prove that $\text{rank}(M_t) \leq \lambda_{s,t}$ with high probability. The plan is to show that the global encoding vector on each incoming edge of t is a linear combination of the global encoding vectors in a minimum s-t cut with high probability. Consider a minimum s-t cut $\delta^{in}(T)$ where $T \subset V$ with $s \notin T$ and $t \in T$ and $d^{in}(T) = \lambda_{s,t}$. Let $E' = \{e'_1, \ldots, e'_{m'}\}$ be the set of edges in E with their heads in T. Let $\lambda = \lambda_{s,t}$ and assume $\delta^{in}(T) = \{e'_1, \ldots, e'_\lambda\}$. See Figure 3.2a for an illustration. Let F' be the $d \times m'$ matrix $(f_{e'_1}, \ldots, f_{e'_{m'}})$. Let K' be the $m' \times m'$ submatrix of K restricted to the edges in E'. Let H' be the $d \times m'$ matrix $(f_{e'_1}, \ldots, f_{e'_\lambda}, \vec{0}, \ldots, \vec{0})$. Then, by the network coding requirements, the matrices satisfy the equation

$$F' = F'K' + H'.$$

By the same argument as in Lemma 3.7, the matrix $(I - K')$ is nonsingular with probability at least $1 - O(1/m^{c+2})$. So the above matrix equation can be rewritten as $F' = H'(I - K')^{-1}$. This implies that every global encoding vector in F' is a linear combination of the global encoding vectors in H', which are the global encoding vectors in the cut $\delta^{in}(T)$. Therefore the $\text{rank}(M_t) \leq d^{in}(T) = \lambda_{s,t}$.

Now we prove that $\text{rank}(M_t) \geq \lambda_{s,t}$ with high probability. The plan is to show that there is a $\lambda \times \lambda$ submatrix M'_t of M_t such that $\det(M'_t)$ is a non-zero polynomial of the local encoding coefficients with small total degree. First we use the edge disjoint paths from s to t to define M'_t. Let $\lambda = \lambda_{s,t}$ and $P_1, \ldots, P_\lambda$ be a set of λ edge disjoint paths

from s to t. Set $k_{e',e} = 1$ for every pair of adjacent edges $e', e \in P_i$ for every i, and set all other local encoding coefficients to be zero. See Figure 3.2b for an illustration. Then each path sends a distinct unit vector of the standard basis to t, and thus M_t contains a $\lambda \times \lambda$ identity matrix as a submatrix. Call this $\lambda \times \lambda$ submatrix M'_t. Next we show that the $\det(M'_t)$ is a non-zero polynomial of the local encoding coefficients with small total degree. Recall that $F = H(I-K)^{-1}$. By considering the adjoint matrix of $I-K$, each entry of $(I-K)^{-1}$ is a polynomial of the local encoding coefficients with total degree m divided by $\det(I-K)$, and thus the same is true for each entry of F. Hence $\det(M'_t)$ is a degree λm polynomial of the local encoding coefficients divided by $(\det(I-K))^{\lambda}$. By using the edge disjoint paths $P_1, \ldots, P_\lambda$, we have shown that there is a choice of the local encoding coefficients so that $\text{rank}(M'_t) = \lambda$. Thus $\det(M'_t)$ is a non-zero polynomial of the local encoding coefficients. Conditioned on the event that $\det(I-K)$ is nonzero, the probability that $\det(M'_t)$ is zero is at most $O(\lambda/m^{c+2}) \leq O(1/m^{c+1})$ by the Schwartz-Zippel Lemma. By bounding the probability that $\det(I-K) = 0$ using Lemma 3.7, we obtain that the probability that $\det(M'_t) = 0$ is at most $O(1/m^{c+1})$. This implies that M'_t is a full rank submatrix with high probability, and thus $\text{rank}(M_t) \geq \text{rank}(M'_t) = \lambda$. We conclude that $\text{rank}(M_t) = \lambda_{s,t}$ with probability at least $1 - O(1/m^{c+1})$ by the union bound. □



(a) Upper bound by min-cut.

(b) Lower bound by paths.

Figure 3.2

Thus the probability that $\text{rank}(M_t) \neq \lambda_{s,t}$ for some $t \in V - s$ is at most $n \cdot O(1/m^{c+1}) \leq O(1/m^c)$ by the union bound on t, and this proves the second part of Theorem 3.5. Therefore, we only need to set c to be a constant to guarantee a high probability result, while each field operation can be done in $O(\log m)$ time.

3.3 Connectivities in Acyclic Graph

In directed acyclic graphs, one can compute the global encoding vectors directly by following the topological ordering of the graph. We can first preprocess the graph by a breadth first search to remove all vertices that are not connected from the source vertex. Then the source vertex s is the only vertex with indegree zero, and we set the global encoding vectors of its outgoing edges to be the vectors in the standard basis, as required by the first condition for the network coding solution. We assign random local encoding coefficients to each pair of adjacent edges. Then we process the remaining vertices following the topological ordering of the graph, so that when each vertex v is processed all the vectors of its incoming edges are already computed. For each outgoing edge of v, we compute its vector by taking a linear combination of the vectors of the incoming edges of v, according to the local encoding coefficients. It is easy to see that the global encoding vectors of all edges will be computed, and the resulting vectors satisfy the second requirement of the network coding solution.

Let d be the outdegree of the source vertex s. So each global encoding vector is of dimension d. For a vertex v with indegree $d^{in}(v)$, it requires $d \cdot d^{in}(v)$ arithmetic operations to compute the vector of one outgoing edge of v, and thus it requires $d \cdot d^{in}(v) \cdot d^{out}(v)$ operations to compute the vectors of all outgoing edges of v. Therefore the total encoding time of this straightforward implementation is $\sum_{v \in V-s} d \cdot d^{in}(v) \cdot d^{out}(v)$, and in worst case it requires $\Theta(dnm)$ steps since $\sum_{v \in V-s} d^{in}(v) \cdot d^{out}(v) \leq n \sum_{v \in V-s} d^{out}(v) = \Theta(nm)$.

In this section we are going to obtain faster encoding algorithms. By improving the algorithm we can compute single source all pairs connectivities faster. In the remaining of this section we will discuss how we can improve by applying fast matrix multiplications. Next we will state an important transformation of the graph using superconcentrators so that the connectivities are preserved in the transformed graph while it has constant

indegree. This transformation provides faster algorithms for encoding.

3.3.1 Faster Encoding Algorithms

An improvement is to use a fast rectangular matrix multiplication algorithm to compute all the outgoing vectors of a vertex. First we observe that for every vertex $v \neq s$, it takes $O(d \cdot d^{in}(v)d^{out}(v))$ time to compute all global encoding vectors that are on the edges of outgoing from v, using simple matrix multiplication. As discussed in Section 2.3.1.1, we can perform the multiplication in $O(d \cdot (d^{out}(v))^{\omega-2}d^{in}(v))$ time. Hence the total time of encoding is given by

$$\begin{aligned}
\sum_{v \in V-s} d \cdot (d^{out}(v))^{\omega-2}d^{in}(v) &= d \sum_{v \in V-s} (d^{out}(v))^{\omega-2}d^{in}(v) \\
&\leq dn \sum_{v \in V-s} (d^{out}(v))^{\omega-2} \\
&\leq dn\left(\sum_{v \in V-s} d^{out}(v)\right)^{\omega-2} \qquad \text{(By Lemma 3.19)} \\
&= O(dnm^{\omega-2}).
\end{aligned}$$

This already gives a faster algorithm than the naïve approach since $\omega < 3$.

Next we consider an efficient implementation of this algorithm using superconcentrators.

Theorem 3.9. *Given a simple directed acyclic graph and a source vertex s with outdegree d, the encoding step can be done in $O(dm)$ time. In addition, the edge connectivities from s to all vertices can be computed in $O(d^{\omega-1}m)$ time.*

Proof. Observe that the encoding operation is much faster if the indegree of a vertex is a constant. This is because encoding any vertex $v \neq s$ amounts to multiplying a $d \times d^{in}(v)$ matrix with a $d^{in}(v) \times d^{out}(v)$ matrix, so that if both $d^{in}(v)$ and $d^{out}(v)$ are constant, then encoding v takes $O(d)$ time by simple matrix multiplication.

So the idea is to transform the graph into a bounded degree graph, and it turns out that superconcentrators give us an optimal transformation for this purpose. To improve

the encoding time, we first transform the graph $G = (V, E)$ by replacing each vertex in $V - s$ by a superconcentrator (see Section 2.1.9 for more details). For each vertex v, let $d_v = \max\{d^{in}(v), d^{out}(v)\}$, we replace v by a superconcentrator Γ_v with $|I| = |O| = d_v$, and "rewire" each incoming edge of v to a distinct vertex in I and each outgoing edge of v from a distinct vertex in O. See Figure 3.3 for an illustration. The total transformation time is $\sum_{v \in V} O(d_v) = \sum_{v \in V} O((d^{in}(v) + d^{out}(v))) = O(m)$.

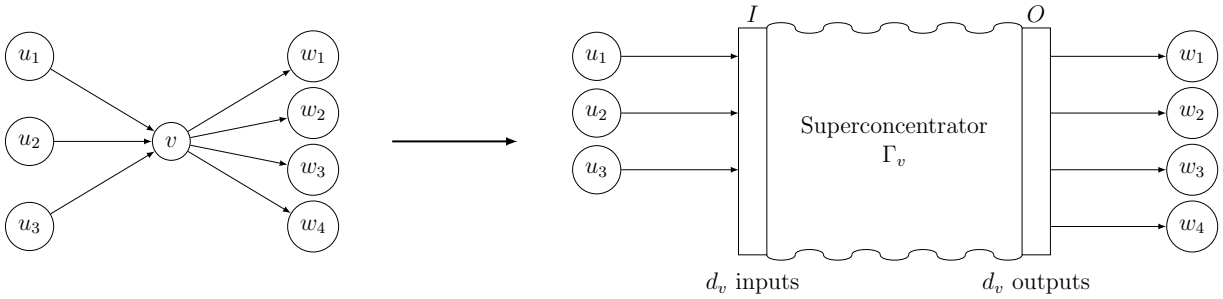


Figure 3.3: Replace vertex v by a superconcentrator Γ_v and "rewire" the edges.

Call the graph after the transformation G'. A key point is that the edge connectivity from the source s to a vertex t in G is equal to the edge connectivity from the source s to Γ_t in G' by thinking of Γ_t as a node. To see this, any set of edge disjoint paths from s to t in G corresponds to a set of edge disjoint paths from s to Γ_t in G', because paths sharing a vertex v in G can be routed using the disjoint paths guaranteed by the superconcentrator Γ_v in G'.

By the properties described in Section 2.1.9, G' is a directed acyclic graph with constant maximum indegree. Thus the global encoding vector of one edge in G' can be computed in $O(d)$ time, where d is the outdegree of the source vertex s. Since Γ_v has $O(d_v)$ edges, the global encoding vectors of the outgoing edges of Γ_v can be computed in $O(d \cdot d_v)$ time. Therefore, all the global encoding vectors can be computed in $\sum_{v \in V} O(d \cdot d_v) = \sum_{v \in V} O(d \cdot (d^{in}(v) + d^{out}(v))) = O(dm)$ time.

To compute the edge connectivity from s to Γ_t in G', by Theorem 3.5 we can compute the rank of the incoming vectors of Γ_t in G'. For a rectangular matrix of size $a \times b$, its rank can be computed in $O(ab^{\omega-1})$ time [32]. So the total decoding time is given by

$$\sum_v O(d^{in}(v) \cdot d^{\omega-1}) = O(m \cdot d^{\omega-1}).$$

□

Note that the encoding time is optimal since writing down all the global encoding vectors already takes $O(dm)$ time. It is surprising that the vectors of all the outgoing edges of Γ_v in G' can be computed in $O(d \cdot d_v)$ time, the same time complexity as computing the vector for just one outgoing edge of v in G. This can be used to improve the random linear coding algorithm [35] for network coding.

Our algorithm is faster than running the Even-Tarjan algorithm for $\Omega(n)$ times for all n and m. When d is small we can compute single source edge connectivites in linear time. Recall from Section 2.1.2 that $\lambda_{s,t}$ is the edge connectivity from s to t. When the outdegree of s is unbounded, this can be used to obtain an algorithm to compute $\min\{\lambda_{s,t}, d\}$ for all t in linear time for constant d, by adding a new source s' with d outgoing edges to s and compute the edge connectivities from s'. This also gives us an efficient linear time checker to test whether the multicasting rate of G is at least d by computing $\min\{\lambda_{s,t}, d\}$ for all t that belong to a set of receivers $T \subseteq V$.

3.4 Directed Planar Graphs

In this section we will discuss how to improve the algorithms for directed planar graphs, bounded genus graphs, and fixed minor free graphs (see Section 2.1.1 for the definitions). Using the matrix formulation in Section 3.2, we could find a network coding solution by solving systems of linear equations.

Claim 3.10. *Finding a network coding solution F amounts to solving d systems of linear equations, where $d = d^{out}(s)$.*

Proof. Since the source vertex has outdegree d, the global encoding vectors are of dimension d (see Section 3.2 for details). The matrix formulation that we obtained in Section 3.2 is equivalent to

$$(I - K)^T F^T = (H_s)^T.$$

Hence we can obtain d systems of linear equations since the matrix form above can be

illustrated as follows,

$$\underbrace{\begin{pmatrix} & a_{e_j,e_1} & \\ & \vdots & \\ \cdots & a_{e_j,e_i} & \cdots \\ & \vdots & \\ & a_{e_j,e_m} & \end{pmatrix}}_{m\times m \text{ matrix}} \underbrace{\begin{pmatrix} & \vdots & \\ - & f_{e_j} & - \\ & \vdots & \end{pmatrix}}_{m\times d \text{ matrix}} = \underbrace{\begin{pmatrix} - & \vec{e}_1 & - \\ & \vdots & \\ - & \vec{e}_d & - \\ - & \vec{0} & - \\ & \vdots & \\ - & \vec{0} & - \end{pmatrix}}_{m\times d \text{ matrix}},$$

where $a_{e_j,e_i} = (I-K)_{e_j,e_i}$. The k-th system of linear equations $Ax_k = b_k$ can be obtained by setting $x_k = (F^T)_{*,k}$, $b_k = ((H_s)^T)_{*,k}$, and $A = (I-K)^T$. □

To solve a system of linear equations in general, one can apply Gaussian elimination on A, and then perform backward substitution, which runs in $O(m^\omega)$ time. However, from Section 2.3.2, we see that if the underlying graph of $(I-K)$ has a weak separator tree, then this system of linear equations can be solved more efficiently using nested dissection.

Assume that the original network G as a simple directed planar graph with constant maximum degree d, and let G' be the underlying graph of $(I-K)$. Observe that G is planar does not imply that G' is planar, because it is obtained from the line graph of a planar graph. See Figure 3.4 for an example.

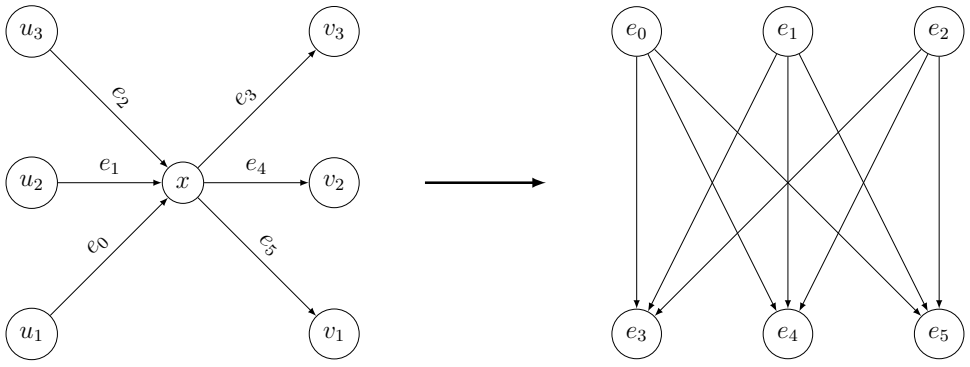


Figure 3.4: An example that shows a planar graph is not necessarily planar in its line graph. In particular, the complete bipartite graph $K_{3,3}$ on the right is the line graph of the left one. The left graph is clearly planar while its line graph is not.

Nevertheless, if d is a constant, then we can argue that G' also has a good separator.

Lemma 3.11. *Given a simple directed planar graph G with constant maximum degree d, its line graph has a $(O(\sqrt{n}), \alpha)$-separation, where $\alpha < 1$.*

Proof. By Theorem 2.4, any planar graph has a $(O(\sqrt{n}), 2/3)$-separator Z. We consider all the directed edges that have at least one endpoint in Z. Denote the set of these edges as Z'. Since the maximum degree of G is d, which is a constant, $|Z'| = O(d\sqrt{n}) = O(\sqrt{n})$. We will take Z' as the separator of the line graph of G, and argue that the two separated parts X' and Y' are of size at most αn, where α is a constant smaller than one.

Without loss of generality, we assume that the graph has no isolated vertices. By Theorem 2.4, $|X \cup Z| \leq 2n/3$ and $|Y \cup Z| \leq 2n/3$, hence we have that both X and Y are of size at least $n/3$. Therefore there are at least $n/3-1$ edges with an endpoint in X, and similarly for Y. Since Z' has at most $O(\sqrt{n})$ edges, the number of directed edges with both endpoints in X is at least $n/3 - 1 - O(\sqrt{n}) = \beta n$. The same argument holds for Y. Since these edges correspond to the vertices in X' and Y', therefore we can conclude that both $|X' \cup Z'|$ and $|Y' \cup Z'|$ are at most $(1-\beta)n + O(\sqrt{n}) = \alpha n$, where α is a constant less than one. This implies that the line graph of G has a $(O(\sqrt{n}), \alpha)$-separation. □

Theorem 3.12. *Given a simple directed planar graph G with constant maximum degree d, the encoding time of G is $O(n^{\omega/2})$.*

Proof. By Lemma 3.11, finding a separator for G' is equivalent to finding a separator for G, which can be done in linear time [52]. Hence we apply Theorem 2.14 with $\gamma = 1$, $\beta = 1/2$ and δ a constant (since the maximum degree is a constant), we obtain an $O(n^{\omega/2})$ time randomized algorithm for solving one system of linear equation $Ax_k = b_k$. By Claim 3.10 there are d systems of linear equations, it takes $O(dn^{\omega/2})$ time to complete the encoding process. However, as d is a constant, the total encoding time is $O(n^{\omega/2})$. □

In order to compute the edge connectivity from s to t, we need to find the rank of a $d^{out}(s) \times d^{in}(t)$ matrix, according to Theorem 3.5. As $d^{out}(s) = d$ is constant and $d^{in}(t) \leq d$, the matrix has constant size in both dimensions. Hence computing $\lambda_{s,t}$ can be done in constant time. Thus we can compute the edge connectivities from a fixed source s efficiently.

Theorem 3.13. *Given G is directed, planar with constant maximum degree d, and a given source s, the edge connectivities from s to all vertices can be computed in $O(n^{\omega/2})$ time.*

Proof. By Theorem 3.12 we can perform encoding for this graph in $O(n^{\omega/2})$ time. For any $t \in V$, its decoding time is constant. Hence decoding all vertices takes $O(n)$ time since $\sum_{t \in V} O(d^{out}(s) \cdot d^{in}(t)) \leq n \cdot O(1) = O(n)$. We finish the proof by observing that the encoding time actually dominates the whole algorithm since $\omega \geq 2$. □

The above results can be applied to other classes of graphs that have good separators. For the class of bounded genus graphs with constant maximum degree, a $(O(\sqrt{n}), 2/3)$-separator can be found in linear time [26]. Hence we can apply the same argument to find a network coding solution in $O(n^{\omega/2})$ time.

Corollary 3.14. *Given G is a bounded genus graph with constant maximum degree and a source s, the edge connectivities from s to all vertices can be computed in $O(n^{\omega/2})$ time.*

For the class of fixed minor free graphs with constant maximum degree, it is known that there is an $O(n^{1.5})$ time algorithm [4] to find a $(O(\sqrt{n}), 2/3)$-separator. Apply Theorem 2.14 we obtain the following corollary.

Corollary 3.15. *Given a fixed minor free graph G with constant maximum degree and a source vertex s, the edge connectivities from s to all vertices can be computed in $O(n^{1.5})$ time.*

Alon and Yuster [5] showed how to improve the time complexity to $O(n^{3\omega/(3+\omega)}) = O(n^{1.33})$, by using a faster algorithm to find a larger separator (so γ is smaller but β is bigger), and this improved algorithm can be used for computing single source edge connectivities as well. Very recently, Kawarabayashi and Reed [44] proposed an $O(n^{1+\epsilon})$ time algorithm for finding a separator in fixed minor-free graphs for any $\epsilon > 0$, and this can be used to improve the running time.

Corollary 3.16. *Given a fixed minor free graph G with constant maximum degree and a source vertex s, the edge connectivities from s to all vertices can be computed in $O(n^{\omega/2})$ time.*

3.5 All Pairs Edge Connectivities

To solve $F = FK + H_s$ for the global encoding vectors, one can compute the inverse of $I - K$ and get $F = H_s(I - K)^{-1}$. It takes $O(m^\omega)$ time to compute $(I - K)^{-1}$ since the matrix $(I - K)$ is of size $m \times m$, but we observe that all pairs edge connectivities can be computed readily once we have $(I - K)^{-1}$.

Lemma 3.17. *For any source vertex $s \in V$, the network coding solution F can be determined by taking a submatrix of $(I - K)^{-1}$ directly. In particular, $F = ((I - K)^{-1}))_{\delta^{out}(s),*}$.*

Proof. In our setup, H_s is a $d^{out}(s) \times m$ matrix with a $d^{out}(s) \times d^{out}(s)$ identity matrix in the columns corresponding to $\delta^{out}(s)$. So F is just equal to $((I - K)^{-1})_{\delta^{out}(s),*}$ up to permuting rows. □

Therefore $(I - K)^{-1}$ contains all the global encoding vectors for all source vertices, and thus the total encoding time for all pairs is $O(m^\omega)$.

To compute the edge connectivity from s to t, by Theorem 3.5 we compute the rank of $F_{*,\delta^{in}(t)}$ and this is just equal to the rank of $((I - K)^{-1})_{\delta^{out}(s),\delta^{in}(t)}$.

Theorem 3.18. *Given a vertex s, computing the ranks of $((I - K)^{-1})_{\delta^{out}(s),\delta^{in}(t)}$ for all t can be done in $O(m \cdot (d^{out}(s))^{\omega-1})$ time.*

Proof. Since the matrix is of size $d^{out}(s) \times d^{in}(t)$, computing the rank takes $O(d^{in}(t) \times d^{out}(s)^{\omega-1})$ time. Hence for all $t \in V$, the total time of computing the ranks is given by

$$\sum_{t \in V} O(d^{in}(t) \cdot d^{out}(s)^{\omega-1}) = O(m \cdot d^{out}(s)^{\omega-1}).$$

□

To prove the time bound for all pairs edge connectivities, we need the following lemma.

Lemma 3.19. *Let $\{b_1, \cdots b_n\}$ be a set of positive integers, and $k \geq 1$. Then*

$$\sum_{i=1}^{n} b_i^k \leq \left(\sum_{i=1}^{n} b_i \right)^k$$

Proof. Let $B = \sum_{i=1}^{n} b_i$. Obviously we have $\frac{b_i}{B} \leq 1 \leq b_i$ for all i. Then,

$$\sum_{i=1}^{n} \left(\frac{b_i}{B}\right)^k \leq 1$$
$$\sum_{i=1}^{n} b_i^k \leq B^k$$
$$\sum_{i=1}^{n} b_i^k \leq \left(\sum_{i=1}^{n} b_i\right)^k$$

□

Theorem 3.20. *All pairs edge connectivities can be computed in $O(m^\omega)$ time.*

Proof. By Theorem 3.18 and summing over all possible choices of source vertex s, the total decoding time is $\sum_{s \in V} O(m \cdot (d^{out}(s))^{\omega-1})$. Since $d^{out}(s) \leq n$ in a simple graph, by Lemma 3.19, this sum is at most $O(m^\omega)$. Since encoding can be done in $O(m^\omega)$ time, computing all pairs edge connectivities can be done in the same time bound. □

In fact a tighter bound on decoding can be obtained. This will be useful for graphs that have a faster encoding algorithm, as we will describe below.

Theorem 3.21. *The total decoding time is bounded by $O(m^2 n^{\omega-2})$.*

Proof. Consider $D = \{d^{out}(s) : s \in V\}$. We pick any two distinct elements d_i, d_j from D such that $d_i + d_j \leq n$. Replace D by removing d_i, d_j and insert $d_i + d_j$. Repeat until we cannot pick any such pair. It is obvious to see that there is at most one element d_i in D such that $d_i \leq \frac{n}{2}$. Hence we have $|D| = O\left(\frac{m}{\frac{n}{2}}\right) = O(\frac{2m}{n})$. Since each element in D is at most n, by Lemma 3.19, we can conclude that

$$\sum_{s \in V} O(m \cdot (d^{out}(s))^{\omega-1}) = O(m \cdot \frac{2m}{n} \cdot n^{\omega-1})$$
$$= O(m^2 n^{\omega-2}).$$

□

Our algorithm is faster than running the Even-Tarjan algorithm for $\Omega(n^2)$ pairs as long as $m = O(n^{1.93})$. For $m = O(n)$ our algorithm runs in $O(n^\omega)$ time while running

the Even-Tarjan algorithm for $\Omega(n^2)$ pairs takes $O(n^{3.5})$ time.

It turns out that if the given graph has good separator (see Section 2.1.7 for more details), we can compute $(I-K)^{-1}$ faster. Combining with Theorem 3.21, we can compute all pairs edge connectivities faster in these graphs. The following result is proven in [15].

Theorem 3.22 ([15]). *All pairs edge connectivities can be computed in $O(d^{\omega-2}n^{\omega/2+1})$ time for any directed fixed minor free graph with maximum degree d.*

3.5.1 Connections with Previous Work

We note that similar work has been done by Cheriyan [14], who obtained an algebraic formulation to compute s-t vertex connectivities.

Theorem 3.23 ([14], Theorem 4.3). *Let $G = (V, E)$ be a directed graph, and $s, t \in V$ with $s \neq t$. Let Q be an $n \times n$ matrix such that*

$$Q_{i,j} = \begin{cases} q_{i,i} & \text{if } i = j \\ q_{i,j} & \text{if } i \neq j \text{ and } (i,j) \in E \\ 0 & \text{otherwise} \end{cases},$$

where each $q_{i,j}$ is a distinct indeterminate. Let $X = \{x : (s,x) \in E\}$ and $Y = \{y : (y,t) \in E\}$. By randomly assigning each indeterminate with an integer from $\{1, \ldots W\}$, then with probability at least $1 - n/W$, Q is non-singular. If Q is non-singular, then with probability at least $1 - n^2/W$, the s-t vertex connectivity equals to $\text{rank}((Q^{-1})_{X,Y})$.

We are going to show that the method used in this section can be derived from Cheriyan's work and vice versa. We first show how Theorem 3.23 derives our formulation. Given a directed graph $G = (V, E)$, and two distinct vertices $s, t \in V$, we construct the line graph L_G of G. Then for each outgoing edge of s in G, we add a super source s' and connect an edge to the associated vertex in L_G. We do the same to connect each vertex in L_G which corresponds to an incoming edge of t in G to a super sink t'. Now it is clear that the s-t edge connectivity in G is equal to the s'-t' vertex connectivity in L_G. By Theorem 3.23, Q is an $m \times m$ matrix, as L_G has m vertices. It can be checked that

Q is identical to $I - K$ in our formulation by associating $Q_{i,j}$ with $K_{i,j}$ for $i \neq j$, and assigning $Q_{i,i} = 1$.

Next we show that Cheriyan's formulation can be derived from our work. Given a directed graph G, we wish to compute the vertex connectivity between s and t. Using Theorem 3.23 we just need to compute the rank of the inverse of an $n \times n$ matrix. In general our formulation deals with a matrix with larger size, but we can use our formulation to derive the more compact one in Theorem 3.23. Note that computing the s-t vertex connectivity in G is equivalent to computing the s-t edge connectivity in a directed graph G' by a simple transformation, which is discussed in Section 2.1.3. Every vertex in G' has indegree or outdegree exactly equal to one. Then every vertex in G' either transmits a single random linear combination of vectors to the only outgoing edge, or simply forwards the only incoming vector to all its outgoing edges. In either case we can simply store the global encoding vector in each vertex in G instead of every edge. Then we can obtain the formulation with respect to G below from (3.1):

$$\begin{pmatrix} & | & \\ \dots & f_v & \dots \\ & | & \end{pmatrix} = \begin{pmatrix} & | & \\ \dots & f_v & \dots \\ & | & \end{pmatrix} \begin{pmatrix} & \vdots & \\ \dots & k_{i,j} & \dots \\ & \vdots & \end{pmatrix} + \begin{pmatrix} & | & & | & \\ \dots & \vec{e_i} & \dots & \vec{0} & \dots \\ & | & & | & \end{pmatrix}.$$

Now K is an $n \times n$ matrix such that

$$K_{i,j} = \begin{cases} k_{i,j} & \text{where } (i,j) \text{ is an edge in } G \\ 0 & \text{otherwise} \end{cases},$$

where each $k_{i,j}$ is a distinct indeterminate. Then by Theorem 3.5 and Lemma 3.17, the s-t edge connectivity in G' can be computed by $\text{rank}(((I-K)^{-1})_{\delta^{out}(s),\delta^{in}(t)})$. It matches the formulation in Theorem 3.23, except that the diagonal entries are different.

3.6 Edge Splitting-off

In this section we show that expanders and superconcentrators can be applied to design fast algorithms for another well-studied graph connectivity problem.

Consider the edge splitting-off problem in Section 2.1.6. Recall that splitting off a pair of edges (xu, xv) in an undirected graph $G = (V, E)$ means deleting these two edges and adding a new edge uv if $u \neq v$. There are many applications of graph connectivity problems using edge splitting-off. We refer readers to Section 2.1.6 for more details.

In this section, our goal is to design a fast algorithm to completely split-off x (see Section 2.1.6 for the definition). A straightforward algorithm is to try every pair of edges on x, and check whether the edge connectivities decrease for some pairs after its splitting-off. In the worst case this requires $O((d(x))^2)$ attempts to completely split-off x. We show how to use expanders and superconcentrators to completely split-off x in $O(d(x))$ attempts in undirected graphs and $O(d^{in}(x))$ attempts in Eulerian directed graphs respectively.

3.6.1 Edge Splitting-off in Directed Graphs

Recall that Theorem 2.3 states that in the Eulerian directed graph, there exists an edge pair to be split-off for every vertex. However it is not known how to completely split-off a vertex faster than $O((d^{in}(x))^2)$ attempts, and we show how to do it in $O(d^{in}(x))$ attempts using superconcentrators. Superconcentrators can be used to reduce the maximum degree significantly while preserving the edge connectivities and only increasing the number of vertices moderately. We replace vertex x by a superconcentrator Γ_x with $O(d^{in}(x))$ inputs and $O(d^{in}(x))$ outputs as in Figure 3.3.

Theorem 3.24. *Let $G = (V, E)$ be a directed graph. Let G_x be the directed graph obtained by replacing a vertex x in G with a superconcentrator Γ_x with $O(d^{in}(x))$ inputs and $O(d^{in}(x))$ outputs. Then for every pair of vertices $s, t \in V - x$, $\lambda_{s,t}$ in G is the same as that in G_x.*

Proof. Assume that there are $k \leq \lambda_{s,t}$ edge disjoint paths connecting from s to t which pass through vertex x in G. Let S and T be the set of incoming and outgoing edges of vertex x that are occupied by those k edge disjoint paths, and $|S| = |T| = k$. Now each of these edges in G_x are connected to k inputs and outputs of Γ_x. By the definition of superconcentrator, there are k vertex-disjoint paths from the k inputs to the k outputs. Hence we see that the s-t edge connectivity in G_x is at least $\lambda_{s,t}$. □

Theorem 3.24 implies that completely split-off x in G is equivalent to completely split-off every vertex in Γ_x. However not all vertices in Γ_x have the same indegree and outdegree. We can add extra edges within Γ_x to make sure that every vertex has the same indegree and outdegree and Γ_x becomes an Eulerian directed graph. Then by Theorem 2.3, we can completely split-off every vertex in Γ_x.

Theorem 3.25. *Given a directed graph $G = (V, E)$, completely split-off a vertex x can be done in $O(d^{in}(x))$ attempts.*

Proof. Construct the graph G_x as in Theorem 3.24. By the definition of superconcentrator, each vertex in Γ_x has constant degree, therefore it can be completely split-off in $O(1)$ attempts. Since there are $O(d^{in}(x))$ vertices in Γ_x, completely split-off every vertex in Γ_x can be done in $O(d^{in}(x))$ attempts. By Theorem 3.24, after we completely split-off all vertices in Γ_x, this is the same as completely split-off x in G and the edge connectivity between every pair of vertices is preserved. □

3.6.2 Edge Splitting-off in Undirected Graphs

In undirected graphs, it is proved by Lau and Yung [48] that $O(d(x))$ attempts are enough to completely split-off x, using a structural theorem of mincuts. Following the idea in directed graph, we want to use an "undirected superconcentrator" so that by replacing each vertex with this gadget, we can also reduce the number of splitting-off attempts of a vertex x to $O(d(x))$. It turns out that expander graphs can be used as a superconcentrator to prove the same result easily. We replace x by a constant degree expander graph H_x with $O(d(x))$ vertices, and "rewire" each edge of x to a distinct vertex in H_x. See Figure 3.5 for an illustration. Let V_x be the set of vertices in H_x.

To ensure that the edge connectivities are preserved after this operation, we require that $d(S) \geq |S|$ for all S with $|S| \leq |V_x|/2$.

Theorem 3.26. *Let $G = (V, E)$ be an undirected graph. Let G' be the new graph obtained by replacing a vertex x with H_x in G. Then for any pair of vertices $s, t \in V - x$, the s-t edge connectivity in G' is preserved.*

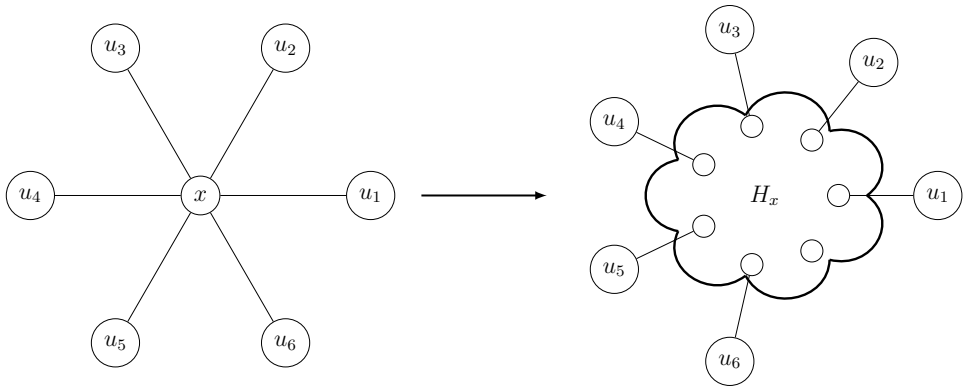


Figure 3.5: Replace vertex x by an expander H_x and "rewire" the edges.

Proof. Assume that there are $k \leq d(x)/2$ edge disjoint paths between s and t that pass through x in G. Then there exist $2k$ edge pairs $(s'_1, t'_1) \ldots (s'_{2k}, t'_{2k})$ such that for every $1 \leq i \leq 2k$, s'_i and t'_i are incident to x, and they are from the same edge disjoint path. Let $S = \{s'_i\}$ and $T = \{t'_i\}$. Now each of the edges in S and T is incident to a distinct vertex in H_x, so we assume that S and T now correspond to the subsets of vertices in H_x. By the property of H_x, $|S| = |T| = k \leq d(x)/2$, $d(S) \geq |S| = k$ and $d(T) \geq |T| = k$. The size of a minimum cut between S and T is at least k. Hence by Menger's Theorem, there are at least k edge disjoint paths between S and T in H_x, and thus the s-t edge connectivity is preserved in G'. □

This implies that if we can completely split-off all vertices in H_x, the resulting graph is equivalent to completely split-off x in the original graph. To ensure that vertices in H_x can be completely split-off, by Theorem 2.2 we need to ensure that all vertices in H_x are of even degree. This can be achieved by adding extra edges to any pair of vertices in H_x that are of odd degree. After that every vertex in H_x can be completely split-off by Mader's theorem.

Claim 3.27. *After replacing an even degree vertex x by a constant degree expander H_x such that every vertex in H_x is of even degree, completely split-off all vertices in H_x can be done in $O(d(x))$ attempts.*

Proof. Since every vertex in H_x is of constant degree, we can completely split-off one vertex in H_x in $O(1)$ attempts by the straightforward algorithm, and thus we can completely split-off all vertices in H_x in $O(d(x))$ attempts since H_x has only $O(d(x))$

vertices. □

We have shown that using constant degree expander graph we can reduce the number of splitting-off attempts of x by a factor of $O(d(x))$. It remains to construct such a constant degree expander graph efficiently and deterministically.

Recall that in Section 2.1.8, we have discussed a deterministic construction of a $(O(n), 3^8, 1)$-expander, which can be constructed in $O(n \log n)$ time. It can be used to replace the vertex in the original graph for complete edge splitting-off. This is the "undirected superconcentrator" that we wanted for the complete edge splitting-off algorithm.

In general, expander graphs and superconcentrators can be used to reduce the maximum degree significantly while preserving the edge connectivities and only increasing the number of vertices moderately. This may be used to reduce the running time of an algorithm that has a super-linear dependency on the maximum degree. We believe that these reductions will find further applications for other graph connectivity problems.

Concluding Remarks

In this thesis we propose an algebraic formulation to compute edge connectivity in general directed graph, using random network coding. We also use superconcentrator and expander to improve algorithms in graph connectivity problems. By some simple transformations, we obtain faster algorithms to compute edge connectivities and find edge splitting-off operations to preserve edge connectivities. The technique of applying network coding to compute edge connectivity is very new as well.

A major question is that whether computing single pair edge connectivity can be improved. Even if we assumed the best coefficient of matrix multiplication, our algorithm is much slower than the fastest combinatorial algorithm, if the graph is not sparse. We hope to see a more compact algebraic formulation, so that computing connectivities could be faster than combinatorial algorithms.

Another important problem is that whether our formulation can be easily generalized to capacitated graphs so as to obtain efficient algorithms. In this work we assumed that the graph is uncapacitated. In that case an edge with capacity k is equivalent to k parallel edges in our problem. As our formulation is based on the number of edges, it would be inefficient in our approach unless there exists more compact formulation that can deal with capacitated graph efficiently.

In directed acyclic graph, we described an algorithm to construct the edge disjoint paths using fast inverse update. However we do not have similar algorithm for general graph. It is because our path construction algorithm is based on topological ordering of directed acyclic graph, which may not be well defined in general graph. Since it is more practical to construct the paths, it would be interesting if there exists such an efficient algorithms.

We use superconcentrator and expander as tools to efficiently solve some graph connectivity problems. We use superconcentrator to show that optimal encoding time in directed acyclic graph can be achieved. We also show that superconcentrator and expander can be used to reduce the number of splitting-off attempts. All these improvements are based on simple transformations. In general, expander graphs and superconcentrators can be used to reduce the maximum degree significantly while preserving the edge connectivities and only increasing the number of vertices moderately. This may be used to reduce the running time of an algorithm that has a super-linear dependency on the maximum degree. We believe that these reductions will find further applications for other graph connectivity problems.

Bibliography

[1] R. Ahlswede, N. Cai, S.R. Li, and R.W. Yeung. *Network Information Flow*. IEEE Transactions on Information Theory, 46(4):1204–1216, 2000.

[2] N. Alon. *Eigenvalues and Expanders*. Combinatorica, 6(2):83–96, 1986.

[3] N. Alon, V. D. Milman. λ_1 , *Isoperimetric Inequalities for Graphs, and Superconcentrators*. Journal of Combinatorial Theory, Series B, 38(1):73–88, 1985.

[4] N. Alon, P. Seymour, R. Thomas. *A Separator Theorem for Nonplanar Graphs*. Journal of the American Mathematical Society, 3(4):801–808, 1990.

[5] N. Alon, R. Yuster. *Solving Linear Systems Through Nested Dissection*. Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science, 225–234, 2010.

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

[7] A. A. Benczúr and D. R. Karger. *Augmenting undirected edge connectivity in $O(n^2)$ time*. Journal of Algorithms, 37(1):2–36, 2000.

[8] A. Bhalgat, R. Hariharan, T. Kavitha, and D. Panigrahi. *An $\tilde{O}(mn)$ Gomory-Hu tree construction algorithm for unweighted graphs*. Proceedings of the 39th annual ACM symposium on Theory of computing (STOC), 605–614, 2007.

[9] A. Bhalgat, R. Hariharan, T. Kavitha, and D. Panigrahi. *Fast edge splitting and Edmonds' arborescence construction for unweighted graphs*. Proceedings of the 19th annual ACM-SIAM symposium on Discrete algorithms (SODA), 455-464, 2008.

[10] U. Brandes, D. Wagner. *A linear time algorithm for the arc disjoint Menger problem in planar directed graphs*. Proceedings of the 5th Annual European Symposium on Algorithms, 64-77, 1997.

[11] J.R. Bunch, J.E. Hopcroft. *Triangular Factorization and Inversion by Fast Matrix Multiplication*. Mathematics of Computation, 28:231–236, 1974.

[12] Y.H. Chan, W. S. Fung, L.C. Lau and C.K. Yung. *Degree Bounded Network Design with Metric Costs*. Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science, 125-134, 2008.

[13] E. Cheng and T. Jordán. *Successive edge-connectivity augmentation problems*. Mathematical Programming, 84(3):577-593, 1999.

[14] J. Cheriyan. *Randomized $\tilde{O}(M(|V|))$ Algorithms for Problems in Matching Theory*. SIAM Journal on Computing, 26(6):1635–1655, 1997.

[15] H. Y. Cheung, L. C. Lau, K. M. Leung. *Graph Connectivities, Network Coding, and Expander Graphs*. To appear in the Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS), 2011.

[16] D. Coppersmith, S. Winograd. *Matrix multiplication via arithmetic progressions*. Journal of Symbolic Computation, 9:251-280, 1990.

[17] J. Dodziuk. *Difference equations, isoperimetric inequality and transience of certain random walks*. Transactions of the American Mathematical Society, 284(2):787–794, 1984.

[18] E. Erez, M. Feder. *Convolutional network codes*. Proceedings of the IEEE International Symposium on Information Theory (ISIT), 146, 2004.

[19] S. Even, R.E. Tarjan. *Network Flow and Testing Graph Connectivity*. SIAM Journal on Computing, 4(4), 507-518, 1975.

[20] A. Frank. *Augmenting graphs to meet edge-connectivity requirements*. SIAM Journal on DiscreteMathematics, 5(1):25–53, 1992

[21] A. Frank and Z. Király. *Graph orientations with edge-connection and parity constraints*. Combinatorica, 22(1):47-70,2002.

[22] O. Gabber, Z. Galil. *Explicit Constructions of Linear-Sized Superconcentrators*. Journal of Computer and System Sciences, 22:407–420, 1981.

[23] H. N. Gabow. *Efficient Splitting Off Algorithms for Graphs*. Proceedings of the 26th annual ACM symposium on Theory of computing, 696–705, 1994.

[24] M. R. Garey, D.S. Johnson. *Computers and Intractability*. Freeman, New York, NY, 1979.

[25] A. George. *Nested dissection of a regular finite element mesh*. SIAM Journal on Numerical Analysis, 10 (2): 345–363, 1973.

[26] J.R. Gilbert, J.P. Hutchinson, and R.E. Tarjan. *A Separator Theorem for Graphs of Bounded Genus*. Journal of Algorithms, 5:391–407, 1984.

[27] J.R. Gilbert, R.E. Tarjan. *The analysis of a nested dissection algorithm*. Numerische Mathematik 50 (4): 377–404, 1987

[28] M.X. Goemans and D.J. Bertsimas. *Survivable networks, linear programming relaxations and the parsimonious property*. Mathematical Programming, 60(1):145-166, 1993.

[29] A.V. Goldberg, S. Rao. *Flows in undirected unit capacity networks*. Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS), 32-34, 1997.

[30] R.E. Gomory, T.C. Hu. *Multi-terminal network flows*. Journal of the Society for Industrial and Applied Mathematics, 9:551–570, 1961.

[31] N. Harvey. *Matchings, Matroids and Submodular Functions*. Ph.D. thesis, Massachusetts Institute of Technology, 2008.

[32] N. Harvey. *Algebraic Algorithms for Matching and Matroid Problems.* SIAM Journal on Computing, 39:679-702, 2009.

[33] N. J. Harvey, D. R. Karger, K. Murota. *Deterministic network coding by matrix completion.* Proceedings of the 16th ACM-SIAM SODA, pp. 489-498, 2005.

[34] T. Ho, B. Leong, R. Koetter, M. Médard, D. R. Karger, M. Effros. *The Benefits of Coding over Routing in a Randomized Setting.* IEEE International Symposium on Information Theory, 2003.

[35] T. Ho, M. Médard, R. Koetter, D.R. Karger, M. Effros, J. Shi, B. Leong. *A Random Linear Network Coding Approach to Multicast.* IEEE Transactions on Information Theory 52, 4413–4430, 2006.

[36] S. Hoory, N. Linial, A. Wigderson. *Expander Graphs and their Applications.* Bulletin (New series) of the American Mathematical Society, 43:439-561, 2006.

[37] J. Hopcroft, R.M. Karp. *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs.* SIAM Journal on Computing, 2:225-231, 1973.

[38] S. Jaggi. *Design and analysis of network codes.* PhD thesis, California Institute of Technology, 2006.

[39] S. Jaggi, P. Sanders, P.A. Chou, M. Effros, S. Egner, K. Jain, L.M.G.M. Tolhuizen. *Polynomial time algorithms for multicast network code construction.* IEEE Transactions on Information Theory 51, 1973–1982, 2005.

[40] K. Jain, M. Mahdian, and M.R. Salavatipour. *Packing Steiner trees.* 14th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2003.

[41] C. Jordan. *Sur les assemblages de lignes.* J. Reine Angew. Math. 70, 185-190, 1869.

[42] T. Jordán. *On minimally k-edge-connected graphs and shortest k-edge-connected Steiner networks.* Discrete Applied Mathematics, 131(2):421–432, 2003.

[43] D.R. Karger, M.S. Levine. *Finding maximum flows in undirected graphs seems easier than bipartite matching.* Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC), 69-78, 1998.

[44] K. Kawarabayashi, B. Reed. *A separator theorem in minor-closed classes.* Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science, 2010.

[45] R. Koetter, M. Médard. *An algebraic approach to network coding.* IEEE/ACM Transactions on Network, 11(5):782–795, 2003.

[46] M. Langberg, A. Sprintson, J. Bruck. *Network coding: A computational perspective.* Proceedings of the 40th Conference on Information Sciences and Systems, 2006.

[47] L.C. Lau. *An approximate max-Steiner-tree-packing min-Steiner-cut theorem.* Combinatorica, 27(1):71-90, 2007.

[48] L.C. Lau, C.K. Yung. *Efficient edge splitting-off algorithms maintaining all-pairs edge-connectivities.* Proceedings of the 14th International Conference on Integer Programming and Combinatorial Optimization (IPCO), 96–109, 2010.

[49] S.-Y.R. Li, R.W. Yeung. *On Convolutional Network Coding.* Proceeding of the 2006 IEEE International Symposium on Information Theory (ISIT 06), 1743–1747, 2006.

[50] S.-Y.R. Li, R.W. Yeung, and N. Cai. *Linear Network Coding.* IEEE Transactions on Information Theory, 49(2):371–381, 2003.

[51] R.J. Lipton, D.J. Rose, and R.E. Tarjan. *Generalized Nested Dissection.* SIAM Journal on Numerical Analysis, 16(2):346–358, 1979.

[52] R.J. Lipton, R.E. Tarjan. *A separator theorem for planar graphs.* SIAM Journal on Applied Mathematics, 36:177–189, 1979.

[53] W. Mader. *A reduction method for edge-connectivity in graphs.* Annals of Discrete Mathematics, 3:145–164, 1978.

[54] K. Menger. *Zur Allgemeinen Kurventheorie*. Fundamenta Mathematicae, 10:95–115, 1927.

[55] M. Mucha, P. Sankowski. *Maximum matchings via Gaussian elimination*. Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 248–255, 2004.

[56] M. S. Pinsker. *On the complexity of a concentrator*. The 7th International Telegraffic Conference, pp 318–318, 1973.

[57] S. Rudich, A.Wigderson. *Computational Complexity Theory*. AMS Bookstore, 2004.

[58] A. Schrijver. *Combinatorial Optimization*. Springer, 2003.

[59] A. Selberg. *On the Estimation of Fourier Coefficients of Modular Forms*. Proceedings of Symposia in Pure Mathematics, Vol. VIII, pp. 1–15, American Mathematical Society, 1965.

[60] A.L. Toledo, X. Wang. *Efficient multipath in sensor networks using diffusion and network coding*. 40th Annual Conference on Information Sciences and Systems, 2006.

[61] L. Valiant. *On Non-linear Lower Bounds in Computational Complexity*. Proceedings of the 7th Annual ACM Symposium on Theory of Computing (STOC), 1997.