Graph Connectivity: Approximation Algorithms and Applications to Protein-Protein Interaction Networks

by

Suzanne Renick Gallagher

B.A., Smith College, 2003

M.S., University of Colorado, 2007

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

2010

This thesis entitled:
Graph Connectivity: Approximation Algorithms and Applications to Protein-Protein Interaction Networks
written by Suzanne Renick Gallagher
has been approved for the Department of Computer Science

Harold Gabow

Debra Goldberg

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Gallagher, Suzanne Renick (Ph.D., Computer Science)

Graph Connectivity: Approximation Algorithms and Applications to Protein-Protein Interaction Networks

Thesis directed by Harold Gabow and Debra Goldberg

A graph is **connected** if there is a path between any two of its vertices and k-connected if there are at least k disjoint paths between any two vertices. A graph is k-edge-connected if none of the k paths share any edges and k-vertex-connected (or k-connected) if they do not share any intermediate vertices. We examine some problems related to k-connectivity and an application.

We have looked at the k-edge-connected spanning subgraph problem: given a k-edge-connected graph, find the smallest subgraph that includes all vertices and is still k-edge-connected. We improved two algorithms for approximating solutions to this problem. The first algorithm transforms the problem into an integer linear program, relaxes it into a real-valued linear program and solves it, then obtains an approximate solution to the original problem by rounding non-integer values. We have improved the approximation ratio by giving a better scheme for rounding the edges and bounding the number of fractional edges. The second algorithm finds a subgraph where every vertex has a minimum degree, then augments the subgraph by adding edges until it is k-edge-connected. We improve this algorithm by bounding the number of edges that could be added in the augmentation step.

We have also applied the idea of k-connectivity to protein-protein interaction (PPI) networks, biological graphs where vertices represent proteins and edges represent experimentally determined physical interactions. Because few PPI networks are even 1-connected, we have looked for highly connected subgraphs of these graphs. We developed algorithms to find the most highly connected subgraphs of a graph. We applied our algorithms to a large network of yeast protein interactions and found that the most highly connected subgraph was a 16-connected subgraph of membrane proteins that had never before been identified as a module and is of interest to biologists. We also looked

at graphs of proteins known to be co-complexed and found that a significant number contained 3-connected subgraphs, one of the features that most differentiated complexes from random graphs.

Dedication

To my parents, Anne and John Gallagher, for having loved and supported me through 27 years of education, from pre-school all the way through graduate school. I can never thank you guys enough.

Acknowledgements

I would like to thank my advisors Debra Goldberg and Harold Gabow, who have taught me so much over the years, as well as all members of the committee.

My family including my parents Anne and John Gallagher, my sister Molly Gallagher, and my fiancé Wayne Vinson for their support and understanding.

My friends Amanda Pingel and Adam Ramsay.

All members of the Goldberg lab; in particular Todd A. Gibson for always being willing to answer my questions and help me make the transition from graph theory into computational biology.

Contents

Chapter

Tables

Table

Figures

Figure

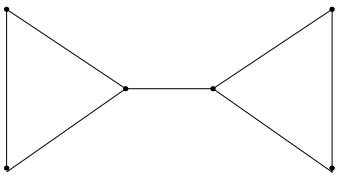
Chapter 1

Introduction

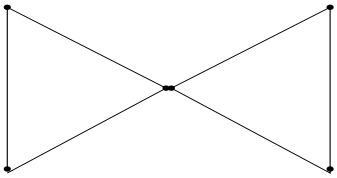
A **graph** is a mathematical structure consisting of vertices and pairs of vertices called edges. Visually, vertices are represented by dots with edges represented by lines between the vertices. In disciplines other than mathematics and computer science, a graph is usually called a **network**, and we will use the two terms interchangeably.

In a graph, there is a path between two vertices if there is a series of edges we can follow to get from the first vertex to the second. A graph is **connected** if there is a path between any two vertices. The idea of connectivity can be extended to k-connectivity for any integer $k > 0$: a graph is k-connected if there are at least k disjoint paths between any two vertices. For a graph to be k-edge-connected, the paths must be edge-disjoint paths, and to be k-vertex-connected (sometimes called k-connected), the paths must not go through any of the same intermediate vertices (note that k-vertex-connected implies k-edge-connected, but not vice versa; see Fig.1.1 for an illustration of the difference). Equivalently, a k-edge-connected graph is one where any $k-1$ edges can be removed and the graph will remain connected. A k-vertex-connected graph is one where any $k-1$ vertices (and all their adjacent edges) can be removed, and the graph will remain connected.

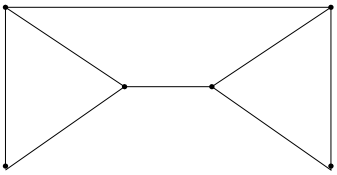
Graph connectivity is closely related to the minimum cut of a graph. In a graph, a cut is a partition of the vertices into two distinct sets. The number of edges with one endpoint on each side of the cut is the value of the cut (these edges are said to cross the cut), and a minimum cut is a cut with the smallest possible value. By Menger's theorem, a graph is k-edge-connected if and only if its minimum cut has value at least k [2]. A vertex cut can be defined as a set of vertices



(a) This graph is connected, but not 2-edge-connected or 2-vertex-connected. Removing the middle edge or either of its endpoints will disconnect the graph.



(b) A graph that is 2-edge-connected, but not 2-vertex-connected. Removing the middle vertex will disconnect the graph.



(c) A 2-connected graph.

Figure 1.1: Examples of connectivity.

whose removal disconnects the graph, and its value is the number of vertices in the cut. As with edge connectivity, a graph is k-vertex-connected if and only if its minimum vertex cut has value at least k.

Connectivity is closely related to minimum degree (the number of edges of which a vertex is a part), because every vertex in a k-connected graph must have degree at least k. The converse, however, is not true; see Fig.1.1(a) for an example of a graph where every vertex has degree 2, but the graph is not 2-connected. There is little relation between connectivity and edge density (the number of edges divided by the number of possible edges). A k-connected graph can have as few as $\frac{kn}{2}$ edges, where n is the number of vertices in the graph, which gives an edge density of $\frac{k}{n-1}$. Depending on the relative values of k and n, this could be very low. For example, a graph of 1001 and vertices with 5 distinct cycles between them would be a 10-connected graph with an edge density of only .01. Conversely, a graph can have an edge density approaching 1 and not even be 1-connected. For example, an $n-1$-clique and an additional vertex with no edges to the clique would have an edge density of $\frac{n-2}{n}$, but would not be connected.

The important feature of a graph with high connectivity is survivability, sometimes also called stability. A graph with high edge density may have many connections between its vertices, but if it has low connectivity, it is vulnerable. The loss of a single edge may disrupt a critical pathway. Therefore, connectivity is important in networks where the maintenance of pathways in the event of edge loss is necessary or desirable. Some examples of graphs where high connectivity is important include Internet connections, where we don't want the loss of a single connection between servers to cause users to lose access to the Web, or the power grid, where we don't want the city to lose power if a single transformer is overloaded.

One class of graphs where this stability can be important is graphs representing biological data, often called biological networks. Biological networks include metabolic networks that model the various chemical reactions in the cell, genetic co-expression networks that have edges between genes that are expressed together, and protein-protein interaction networks that contain information about which proteins bind to each other. Despite the importance of pathways in these

networks, however, graph connectivity (beyond simply determining whether or not a given network is connected) has rarely been applied to them.

Here, we look at some well-known problems in graph connectivity and algorithms for finding solutions to them. Then, we look at a problem of our own design, finding the most highly connected subgraph of a graph. We apply our algorithm for finding the most highly connected subgraph to biological networks, though the algorithm could also be applied to other types of graphs. We begin with some formal definitions.

1.1 Definitions

Throughout the thesis, we will be looking at a graph G with vertex set V and edge set E. This will be notated as $G = (V, E)$. If there is some ambiguity as to what graph a particular vertex or edge set might belong to, they will be further annotated as V_G and E_G. Let $u, v \in V$ and $e = (u, v) \in E$. Let $|V| = n$ and $|E| = m$.

If the order of vertices in an edge $e \in E$ matters, i.e., $(u, v) \neq (v, u)$, then the edge is **directed**. Otherwise, if $(u, v) = (v, u)$, the edge is **undirected**. Most graphs contain only one type of edge: a **directed graph** contains only directed edges, while an **undirected graph** contains only undirected edges. A graph that contains both types of edges is called a **mixed graph**.

A graph is **simple** if for any two vertices u, v, there can be a maximum of one edge that goes between u and v in an undirected graph, or a maximum of one edge from u to v and one edge from v to u in a directed graph. In addition, a simple graph cannot have any loops, edges of the form (u, u). A graph which allows multiple edges between two vertices is a **multigraph**.

The **degree** of a vertex v in an undirected graph is the number of edges which have v as an endpoint and is denoted $d(v)$. In a directed graph, the **out-degree** of v is the number of edges that have v as the first endpoint, while the **in-degree** of v is the number of edges that have v as their second endpoint.

The **edge-connectivity** between two vertices u, v is the number of edge-disjoint paths between them. The **vertex-connectivity** (sometimes just called the connectivity) between two ver-

tices u, v is the number of vertex-disjoint paths between them (for two paths to be vertex disjoint, they must not pass through any of the same intermediate vertices).

A graph G is **k-edge-connected** if all pairs of vertices have an edge-connectivity of at least k and **k-vertex-connected** (or simply k-connected) if all pairs of vertices have a vertex-connectivity of at least k.

A **subgraph** of a graph G is a graph $H = (V_H, E_H)$ where $V_H \subseteq V_G$ and $E_H \subseteq \{(u, v) \in E_G | u, v \in V_H\}$. If $E_H = \{(u, v) \in E_G | u, v \in V_H\}$, then this is called the subgraph **induced** by V_H or simply an **induced subgraph** and denoted as $G[V_H]$.

1.2 Organization of the Thesis

The first part of the thesis discusses a problem in graph connectivity, the k-edge-connected spanning subgraph (k-ECSS) problem. We analyze two different algorithms for approximating this problem.

Chapter 2 gives an introduction to the k-ECSS problem and an overview of previous work. Section 2.2 gives definitions of critical terms, while the remainder of the chapter gives a summary of previous results that have been achieved.

Chapter 3 describes our results on a linear programming algorithm to solve the k-ECSS problem. Section 3.1 gives a summary of our improvements to the previous algorithm. The remainder of the chapter describes our lower bound examples.

Chapter 4 gives a summary of our improvements to a combinatorial algorithm that solves the same problem. Section 4.1 gives a detailed summary of previously published results. Section 4.2 gives our extension these previous results.

The second part of the thesis contains an application of graph connectivity to a particular type of biological networks, protein-protein interaction (PPI) networks. Biologists may wish to skip Chapters 2-4 and start directly in Chapter 5.

Chapter 5 contains a description of PPI networks. Section 5.1 is a general introduction to PPI networks including their major features, and Section 5.2 is a discussion of previous work done

on PPI networks.

Chapter 6 contains our algorithm for finding the most highly connected subgraph and the results of iteratively applying that algorithm to a PPI network.

Chapter 7 contains an analysis of protein complexes using connectivity with the intention of using connectivity to predict unknown complexes from interaction data. In order to determine if connectivity might be a useful indicator of protein complexes, we conducted a survey of known protein complexes and calculated connectivity as well as other metrics currently used to predict protein complexes. We show that connectivity is one of the properties that most differentiates complexes from random subgraphs.

Chapter 8 gives our conclusions as well as directions for future work on the problems discussed in Chapters 3, 4, 6, and 7.

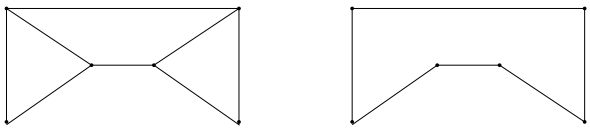
Finally, Appendix A gives full results from Chapter 3. Appendix B contains the additional data from Chapter 7.

Chapter 2

The k-Edge- and k-Vertex-Connected Spanning Subgraph Problem

2.1 Introduction

Suppose we have a k-connected graph G. Is every edge of G necessary to insure that we have k disjoint paths between any two vertices of G? If the answer to that question is "no," then it is reasonable to ask how few edges of the original graph we could keep and still have a k-connected graph. This is the minimum k-connected spanning subgraph problem. The problem is "spanning" because we want to keep all vertices in the original graph and a "subgraph" problem because we are only allowed to use edges that were in the original graph rather than adding additional ones. If we are using edge connectivity, then the problem is called the k-edge-connected spanning subgraph problem and abbreviated k-ECSS; if we are using vertex connectivity, then the problem is simply called the k-connected spanning subgraph problem and abbreviated k-CSS.



(a) A 2-connected graph.

(b) The smallest 2-connected subgraph of the graph in (a).

Figure 2.1: An example of the minimum k-ECSS.

Both problems are specific cases of a much larger problem known as the Generalized Steiner

Network problem. The GSN problem takes as input a complete graph $G = (V, E)$, a cost function on the edges, a set $S \subseteq V$ of vertices that we don't want to see duplicated in distinct u, v paths (other vertices may appear in multiple paths), and a requirement function for the connectivity between each pair of vertices $r : V \times V \to \mathbb{Z}$. The goal is to find a minimum cost set of edges such that all of the connectivity requirements are fulfilled. We can convert the k-ECSS problem into the GSN by making our original graph G into a complete graph G_k by adding edges between all pairs of vertices. Let $S = \emptyset$, $r(u, v) = k$ for all $u, v \in V$, and $c(e) = 1$ if e was an edge in G and $c(e) = \infty$ otherwise. The k-CSS problem uses almost the same conversion, except that $S = V$ rather than being empty. Many other common problems are special cases of the GSN. The simplest of these problems is the minimum cost spanning subtree problem, where our goal is to find the minimum cost connected subgraph of a graph. The Steiner tree and Steiner forest problems, also special cases of the GSN, are similar to the min cost spanning subtree problem, except that only certain vertices are required to be connected. The Steiner tree problem divides the vertices into two categories: required vertices that must be connected in the resulting tree and Steiner vertices which can be part of the tree but are not required to be. The Steiner forest problem is similar, except that there are multiple sets of required vertices that only need to be connected to other vertices in the set. Converting the minimum cost spanning subtree, Steiner tree, and Steiner forest problems into the GSN simply requires setting the appropriate connectivity requirements to 1 and giving illegal edges infinite cost. The Steiner network, or survivable network design problem, includes all edge-connectivity cases of the GSN (i.e., the only restriction from the general case is that $S = \emptyset$). Another special case of the GSN is the graph augmentation problem, where we are given a graph that is not k-connected and want to find the minimum number of edges we would need to add to make it k-connected; this can be converted to the GSN in a similar way to the k-ECSS and k-CSS problems, except that all edges in the original graph have cost 0 and all other edges have cost 1. Various network flow problems, where the goal is to insure a certain amount of "flow" between certain pairs of vertices, also fall under the GSN.

There are many reasons to look at the k-connected and k-edge-connected spanning subgraph

problem. First, they are interesting unsolved problems in graph theory in their own right. Secondly, it is possible that algorithms for solving the k-CSS and k-ECSS problems may be applicable to other GSN problems. Finally, there are many applications of networks in every day life, from computers to roads to biological interactions. In many of these situations, we want a robust network that will be connected even if some number k of our links are unusable.

There are several variations of both types of spanning subgraph problems. Both of these problems have their variants on directed and undirected graphs. The k-ECSS problem also has different variations depending on whether we are looking at simple graphs, which allow for only one edge between any given pair of vertices, or multigraphs, which allow for multiple edges between a given pair of vertices.

The problem of finding the smallest k-edge connected spanning subgraph of a graph G has long been known to be NP-complete for $k \geq 2$ [3]. However, despite the fact that we are unlikely to find an efficient algorithm to find an exact solution, there are several good approximation algorithms. An **approximation algorithm** is an algorithm designed to find the solution to an optimization problem that is within a certain set ratio of the true optimum. Usually, this means a polynomial time approximation algorithm is given for an NP-complete problem. Approximation algorithms are not heuristics which might or might not give a good answer depending on the problem; the performance of an approximation algorithm is guaranteed. Approximation algorithms are also not randomized algorithms; most approximation algorithms are deterministic. Each approximation algorithm has an **approximation ratio**, how close the solution given by the approximation algorithm will be to the true optimum. For example, if we used a $\frac{5}{4}$ approximation algorithm for approximating 2-ECSS on a graph whose minimum 2-ECSS contained 8 edges, our algorithm would be guaranteed to return a 2-ECSS with no more than 10 edges.

2.2 Definitions

All definitions refer to a graph $G = (V, E)$, with vertex set V, edge set E, $u, v \in V$, $e = (u, v) \in E$.

An edge $e = (u, v)$ is a **critical edge** if G is k-connected but $G - e$ is not.

An edge $e = (u, v)$ is a **special edge** if e is a critical edge but neither of its two endpoints have degree exactly k.

The degree of a set of vertices $S \subset V$ in an undirected graph is the number of edges that have exactly one endpoint in S and is denoted as $d(S)$. In a directed graph out-degree of a set of vertices S is the number of edges with only their first endpoint in S, and the in-degree of S is the number of vertices with only their second endpoint in S. For a subset of edges $D \subset E$, let $d_D(v)$ be the degree of v if we count only edges in D, and define $d_D(S)$ for $S \subset V$ analogously to the single vertex case.

A set of vertices in a k-connected graph is **critical** if it has degree exactly k. This is also called a **tight** set. When there is some ambiguity about the value of k, these are referred to as **k-critical sets**.

A set of vertices S in a directed k-connected graph is **in (out) critical** if it has in (out) degree exactly k. A set is **2-way critical** if it is both in and out critical.

A set of vertices S is **bi-critical** if V can be partitioned into three sets S, T_1, T_2 such that T_1 and T_2 are both critical. Analogously, we can define a tri-critical set S, with V partitioned into S, T_1, T_2, T_3. Higher order criticality is defined analogously.

A set of vertices S is **in (out) bi-critical** if V can be partitioned into three sets S, T_1, T_2 such that T_1 and T_2 are both out (in) critical. As with the undirected case, we can also define in or out tri-critical sets, as well as higher order in or out criticalities.

A collection of sets L is called **laminar** if, for any two sets $S, T \in L$, either $S \subset T$, $T \subset S$, or S and T are disjoint.

For a set A and a singleton set $\{s\}$, we will denote $A \cup \{s\}$ as $A + s$.

In a weighted graph, a **fractional edge** is an edge with weight strictly between 0 and 1.

A critical set S in an undirected graph **covers** a critical edge e if e has exactly one endpoint in S. In a directed graph, a set of vertices S **covers** a critical edge $e = (u, v)$ if either S is out-critical and $u \in S, v \notin S$ or S is in-critical and $v \in S, u \notin S$.

2.3 Previous Results

For small values of k, near optimal approximation algorithms for the k-CSS and k-ECSS problems already exist. For $k = 2$, for example, there is a $\frac{5}{4}$ approximation for both the vertex and edge connectivity versions of the spanning subgraph problem[4]. For this reason, our research will be primarily focused on large values of k and exploring trends as k grows larger.

Note that for any k-connected or k-edge-connected graph G, there is an obvious lower bound on the number of edges in the minimum k-CSS or k-ECSS. Because every vertex must have degree at least k, the Handshaking Lemma implies any solution must have a minimum of $\frac{kn}{2}$ edges if G is undirected or kn edges if G is directed. This fact will be of use in exploring many of the following algorithms. This guaranteed lower bound allows us to derive good approximation ratios; in several instances, we will divide all or part of our results by this $\frac{kn}{2}$ lower bound in order to give a good bound.

An equally important, though less obvious result, concerns the upper bound on the number of edges in the solution. By using Menger's theorem and various results on cycles of critical edges, Mader proved that any minimal undirected k-connected graph can have at most kn edges [5]. Mader later achieved a similar result about directed graphs, proving that a minimal k-connected directed graph will have at most $2kn$ edges [6]. (Translation of these results presented in [7]).

There is also an important result about the collection of sets that covers all critical edges. Every critical edge can be covered using a laminar family of tight sets of vertices. The proof of this uses an "uncrossing argument": if we have a laminar family $\mathcal{L}$ of tight sets that does not cover all critical edges, there is another tight set that we can add to $\mathcal{L}$ that increases the number of edges covered without violating laminarity. Take some critical set $S \notin \mathcal{L}$ that covers a critical edge not covered by any of the sets in $\mathcal{L}$. We know that S must exist because there is some critical edge not covered by a set in $\mathcal{L}$, and every critical edge is covered by some critical set; otherwise, the edge could be removed and every cut would still have value at least k, contradicting the fact that the edge is critical. Say that S crosses a set T if $S \not\subset T$, $T \not\subset S$ and $S \cup T \neq \emptyset$. Let the crossing

number of S be the number of sets of $\mathcal{L}$ that S crosses. If the crossing number of S is 0, it can be added to $\mathcal{L}$. Otherwise, there is some set $T \in \mathcal{L}$ that crosses S. Jain showed that $S \cup T$, $S \cap T$, $S - T$, and $T - S$ all have a lower crossing number than S. Further, either $S \cup T$ and $S \cap T$ are both tight and together cover all critical edges covered by S and T, or $S - T$ and $T - S$ are both tight and together cover all critical edges covered by S and T. Thus, one of $S \cup T$, $S \cap T$, $S - T$, or $T - S$ covers a critical edge not covered by any set in $\mathcal{L}$ while crossing fewer sets of $\mathcal{L}$ than S. This argument can be repeated until we have a critical set that covers an additional critical edge and does not cross any sets of $\mathcal{L}$; therefore it can be added to $\mathcal{L}$ without violating laminarity [8, 9].

2.3.1 Linear Programming Algorithms

Linear programming has produced algorithms for finding the smallest k-ECSS with some of the best known approximation ratios to date. Mathematical programming is a technique which seeks to maximize or minimize the value of some function, called the objective function, given a number of constraints, which can be either equations or inequalities. When the objective function and the constraints for a mathematical programming problem are all linear, this is called a linear program, sometimes abbreviated as an LP. If the values of the variables in the linear program must be integers, this is called an integer linear program (ILP) and finding the optimum value for the objective function is NP-Hard. However, if the variables can take real values, then there are several good algorithms for solving linear programs.

The solutions given by linear programming algorithms are called **basic feasible solutions (BFS)**. A basic feasible solution to a linear program with n variables is one that contains at least n constraints that hold with equality and uniquely determine values for the n variables. If the constraints were graphed in n-dimensional space, the solution space of the LP would be represented by an n-dimensional polyhedron with the basic solutions as the corners of the polyhedron. The fact that one of the optimal solutions to a linear program is always a BFS is an important reason that efficient solutions exist to solve linear programs.

The k-ECSS problem can be considered an integer linear program, where our requirement is

that each proper subset of vertices must have in and out degree at least k. Consider a multigraph $G = (V, E)$ where each edge $e \in E$ has multiplicity u_e. Take a vector x with one entry for each edge. Then our goal becomes:

$$\begin{aligned} &\text{minimize} \sum_{e \in E} x_e \\ &\text{subject to :} d_x(S) \geq k && \emptyset \subset S \subset V \\ &x_e \in \{0, 1, \ldots, u_e\} && e \in E \end{aligned}$$

While the number of constraints in this problem is exponential in the number of vertices, Jain showed that there is an equivalent LP with a polynomial number of constraints [8]. For reasons of simplicity, we will continue to show the k-ECSS problem with the notation above, but whenever we talk about "solving" the LP, we will be solving the polynomial version.

As mentioned before, finding an exact solution to this integer linear program would be NP-Hard. However, if we relax the requirements of the integer linear program and allow non-integer values for the edges, then there are efficient algorithms that can solve this problem, giving us a BFS [9]. If we apply this relaxation to our original linear program, we get:

$$\begin{aligned} &\text{minimize} \sum_{e \in E} x_e \\ &\text{subject to :} d_x(S) \geq k && \emptyset \subset S \subset V \\ &0 \leq x_e \leq u_e && e \in E \end{aligned}$$

This relaxation leads to a natural approximation algorithm: solve the relaxation, and for every edge which is included fractionally in the solution, round up to include the full edge.

There are two major advantages to looking at the k-ECSS as a linear program. The linear programming problem has many applications, and so has been studied by mathematicians, computer scientists, and businessmen for many years. As a result, there are efficient algorithms known

for solving it. Linear programs also have a duality result: for every linear program minimization problem, there is a dual maximization problem such that the only solution that the two problems have in common is the optimum. By looking at feasible solutions of the dual problem, we can obtain upper or lower bounds on the solution to the primal problem.

Linear programs have been used successfully in many Generalized Steiner Network Problems. Linear programming gives a 2-approximation for the Steiner tree and Steiner forests problems (where the connectivity requirement between any pair of vertices is either 0 or 1) [10]. Even more important is Jain's proof of a 2-approximation for the edge-connectivity version of the Survivable Network Design Problem (there is an edge connectivity requirement between each pair of vertices) [8]. Jain proves that in any basic feasible solution of the LP, there will be at least one fractional edge f such that the value of f is at least $\frac{1}{2}$. This theorem leads to an iterated algorithm. First, solve the linear program for existing connectivity requirements. Then, for any edge e_i that has a value of 0 or 1, add the constraint $e_i = 0$ or $e_i = 1$ to make sure that edge keeps its current value. Also round up any edge e_f with fractional weight at least $\frac{1}{2}$ and add $e_f = 1$ to the constraints. If there are still some connectivity requirements not satisfied by edges with integer weights, re-solve the linear program with adjusted constraints. Note that in the next stage, the total weight of the edges not constrained to be 0 or 1 cannot increase, because the current fractional weights satisfy all requirements, and therefore a minimum solution cannot have any greater weight. Because of this, the weight of the solution cannot increase by more than the amount that we have rounded. Also note, however, that because of Jain's theorem, at least one fractional edge in the new solution must have weight at least $\frac{1}{2}$, and thus can be rounded in the next stage of rounding. Repeat this procedure until all requirements are satisfied by edges with integer weights, which will take fewer than m iterations. This algorithm gives the desired bound of 2. The solution to the relaxed linear program we solved in the first iteration is less than or equal to the optimum. Because the total weight of the unconstrained edges could not increase and we were only rounding edges with weight at least $\frac{1}{2}$, then through all iterations we can at most double the weights of the fractional edges from the first iteration.

This algorithm gives the desired bound of 2. The solution to the relaxed linear program we solved in the first iteration is less than or equal to the optimum. Because the total weight of the unconstrained edges could not increase and we were only rounding edges with weight at least $\frac{1}{2}$, then through all iterations we can at most double the weights of the fractional edges from the first iteration.

Jain's algorithm has many uses in the general case of the GSN problem. In particular, Jain's algorithm gives the only known constant approximation algorithm for the general case of the undirected Steiner network problem. A modified version of Jain's algorithm can sometimes be used in the directed case: if the requirement function meets certain conditions, then we can prove that there will be a fractional edge f such that the weight of f is at least $\frac{1}{3}$, giving us a 3-approximation algorithm [11]. Unfortunately, this does not work on the general case of the directed Steiner network, which is NP-Hard to approximate within a constant factor.

Gabow, Goemans, Tardos, and Williamson use a linear program for their $1 + \frac{2}{k}$ algorithm for k-connectivity on undirected multigraphs[12]. Their algorithm uses the linear relaxation, then rounds up any fractional edges. In their analysis, they make use of the fact that in a basic feasible solution, the weights of the fractional edges are uniquely determined by a collection of critical sets, meaning that there is a 1-to-1 correspondence between the fractional edges and the sets. As we have already shown, that collection can be laminar, and a laminar family of subsets of an n element set can have no more than $2n$ members. Thus, there are no more than $2n$ fractional edges, and rounding will increase the solution by no more than $2n$. Because $OPT \leq \frac{kn}{2}$ this gives us that the rounded approximation is less than $(OPT + 2n)/OPT \leq 1 + 2n/(kn/2) = 1 + \frac{4}{k}$ times the optimum. Further refinement of this argument can be achieved by looking at the weights of the fractional edges, allowing us to prove a $1 + \frac{2}{k}$ approximation ratio for even values of k and a $1 + \frac{3}{k}$ approximation ratio for odd values of k. If we combine these results with Jain's iterated rounding, this gives a ratio of $1 + \frac{2}{k}$ for all values of k. There exist algorithms for solving linear programs in polynomial time, and because there can be no more than $2n$ fractional edges, the number of iterations of the algorithm is also polynomial. Therefore, the entire algorithm is polynomial, though

it is a large polynomial. Gabow et. al. also showed that the $1 + \frac{2}{k}$ bound is tight for multigraphs by giving an example multigraph on which the linear program will achieve this ratio.

2.3.2 Combinatorial Algorithms

Linear programming algorithms are useful, and they have produced good approximation algorithms, but they have some serious drawbacks. The algorithms used to solve linear programs are expensive in terms of both the time and the space required. Running the iterated rounding algorithm can take as long as $O(n^{10}m^7)$ [8]. It would be far more efficient if, rather than relying on the linear program, we could obtain our approximation algorithm by looking at the combinatorial properties of the graph itself.

Cheriyan and Thurimella developed a combinatorial algorithm to approximate the smallest k-connected subgraph of a simple graph using matching [1]. By using matching, Cheriyan and Thurimella achieve a graph where every vertex has a minimum degree, then augment the graph by adding edges until the graph is k-connected. The approximation ratio of this algorithm is based off of the number of edges that have to be added in the augmentation step. Cheriyan and Thurimella's technique works for both vertex and edge connectivity, although there are slight differences in the way that it is implemented for both. The algorithms achieve a bound of $1+\frac{1}{k}$ for vertex connectivity, $1 + \frac{2}{k+1}$ for edge connectivity in undirected graphs, and $1 + \frac{4}{\sqrt{k}}$ for directed graphs.

For vertex connectivity, the first step is to find the subgraph with the smallest number of edges such that every vertex has degree $k - 1$. This can be done in $O(m^{\frac{3}{2}}(log(n))^2)$ time (it is the b-matching problem, a well-known problem in P). The subgraph is then augmented to make it k-connected. This will produce a graph with no more than $n - 1 + |M|$ edges, where M is the smallest edge set such that every vertex has degree at least $k - 1$. Cheriyan and Thurimella then prove that the optimum k-connected subgraph will have no fewer than $\frac{n}{2} + |M|$ edges. This second step takes $O(m^{\frac{3}{2}}(log(n))^2)$. Combining these two gives the desired approximation ratio of $1 + \frac{1}{k}$ in a time of $O(km^2 + m^{\frac{3}{2}}(log(n))^2)$. This is currently the best known approximation ratio for vertex connectivity [7].

For edge connectivity, two very similar algorithms are used. Variations exist for both the directed and undirected cases. When using the algorithm on edge connectivity, we first find M, a minimum edge set such that every vertex has degree at least k. Again, this can be done in polynomial time. Then, start with $G' = G$ where G is our original graph. For every edge $e \in E - M$, determine if $G' - e$ is k-connected. If it is, remove the edge and continue with $G' = G' - e$. The approximation ratio for this algorithm depends on the number of critical edges from $E - M$, which is less than or equal to the number of special edges in G. For undirected graphs, Cheriyan and Thurimella prove that there will be no more than $\frac{k|V|}{k+1}$ edges in $E - M$ in the final graph by looking at the laminar family of critical sets. This gives an overall approximation bound of $1 + \frac{2}{k+1}$ for the undirected case. For the directed case, we find M as the minimum edge set such that every vertex has in and out degree at least k, then use the same procedure as in the undirected case to augment M into a k-connected graph. Cheriyan and Thurimella prove that there will be no more than $4n\sqrt{k}$ special edges in a directed graph by looking at the laminar families of in- and out-critical sets. This bound on the special edges gives an overall approximation bound of $1 + \frac{4}{\sqrt{k}}$. For both undirected and directed graphs, the time bound is the same as for the vertex connectivity algorithm.

Gabow [13] refines that analysis somewhat. Rather than looking at two families, one of in-critical sets and one of out-critical sets, he looks at a single laminar family that includes both the in- and out-critical sets (the existence of such a set was proved by Frank [14]). For any integer k, Gabow looks at the two unique integers τ and ω such that $0 \leq \omega \leq \tau$ and:

$$k = \frac{\tau(\tau + 1)}{2} + \omega \tag{2.1}$$

To see that these numbers are unique, just note that this equation implies that k is exactly ω larger than the sum of the first τ integers. Using those values for τ and ω, we can define a function, σ defined on k so that:

$$\sigma(k) = \tau + \frac{\omega}{\tau} \tag{2.2}$$

Then, by using certain results about criticality and bi-criticality, Gabow proves that for $k \geq 15$, there will be no more than $n * \sigma(k)$ special edges. This improves the approximation ratio of Cheriyan and Thurimella's algorithm from $1 + \frac{4}{\sqrt{k}}$ to approximately $1 + \frac{\sqrt{2}}{\sqrt{k}}$. Gabow also provides an example which proves that the bound on the number of special edges is tight for all values of k for which it applies.

Multigraphs are also treated in [13]. By improving on the algorithm developed in [15], Gabow is able to use a combinatorial algorithm to find the smallest k-ECSS of a multigraph to within a factor of $2 - \frac{1}{3k}$. At the time, this was the only algorithm for finding the smallest k-ECSS of a directed multigraph that was guaranteed to be under a 2-approximation. A better combinatorial algorithm for undirected multigraphs was developed by Khuller and Raghavachari by using a variant of depth first search, giving a bound of 1.85 [16]. Gabow used the properties of laminar families to refine the analysis and modify the algorithm slightly to give a bound of 1.61 on undirected multigraphs [17]. Currently, all of these bounds have been bested by the LP technique in [12].

2.3.3 Results on Criticality

It is a widely known result mentioned in [1] among others that in simple graphs there are restrictions on the potential size of k-critical sets. A k-critical set can consist of a single vertex, but if there is more than one vertex in the set, then there must be at least k vertices in the set.

Gabow [13] expanded these results to encompass bi-, tri-, and higher order criticality sets as well. For a k-critical set S of order c, $|S| = s$:

$$ck + s(s-1) \geq sk \tag{2.3}$$

When we are dealing with bi-critical sets, we have $c = 2$ and our equation implies that for $k \geq 7$, either $s \leq 2$ or $s \geq k - 1$. For higher order criticalities, similar inequalities will hold, but for

progressively higher values of k.

There are two other implications of the equation that are worthy of note. The first is that either $s \leq c$ or $s \geq k - c + 1$ for values of $c \approx \sqrt{k}$. The second is that a k-critical set of order c can be of size $\frac{c}{2}$ only if $c \geq k/4$.

2.3.4 Theoretical Bounds

As mentioned before, the k-ECSS problem is NP-Complete for $k \geq 2$. A relatively recent result has been to prove that the problem is in fact MAXSNP hard, meaning that there is a limit to how small the approximation guarantee can be for any polynomial algorithm, unless P = NP. For any given instance of the problem, the approximation bound will depend on k.

Fernandes proved that the 2-ECSS problem is MAXSNP-hard for undirected graphs using a reduction from VC3, the vertex cover problem for graphs where every vertex has degree less than or equal to three, to 2-ECSS [18]. The reduction is done by taking an instance G of VC3 and transforming it into an instance H of 2-ECSS. Each edge of G is transformed into a gadget in H consisting of six vertices and six edges; H has one additional vertex where all gadgets are linked together. If we say that G has m edges, n vertices, and a smallest vertex cover C, and H has a smallest 2-ECSS K, it can be shown that:

$$|K| = 6m + |C| \leq 19|C| \tag{2.4}$$

The last inequality comes from the facts that a vertex has degree at most 3, so each vertex in C can cover at most 3 edges, and that C must cover all m edges; thus $3|C| \geq m$. This implies that whatever approximation bound our algorithm gives for 2-ECSS, it will give a better one for VC3, and because VC3 is MAXSNP-hard [19], 2-ECSS will be also.

Gabow et. al. extended this result to cover all values of $k > 2$. Their reduction changes an instance G of VC3 into a multigraph H which then has an k-ECSS K such that if G has m edges and a vertex cover C, then:

$$|K| = (6 + 14k)m + |C| \leq (19 + 42k)|C| \tag{2.5}$$

Given this inequality, for any fixed value of k, the argument is the same as it was for $k = 2$ that k-ECSS is MAXSNP-hard. In fact, this equation implies something more: the approximation bound for any polynomial k-ECSS algorithm will be worse than $1 + \frac{c}{k}$ for some constant c. While the ratio will improve as k gets larger, there is a limit to how fast it can improve.

The results from Gabow et. al. can also be modified to work on digraphs, giving the same approximation results for $k \geq 1$. An analogous result also applies to simple graphs, although this result is not nearly as strong. Because the multigraph has to be broken up and many more vertices have to be added, this technique proves only that the approximation bound for any polynomial k-ECSS algorithm on simple graphs will be worse than $1 + \frac{c}{k^2}$.

Vertex connectivity is also MAXSNP hard. Chumaj and Lingas use a reduction from the traveling salesman problem to prove that there exists some value ϵ such that k-node-connectivity cannot be approximated to within $1 + \epsilon$ [20]. For more general vertex connectivity problems which allow the edges to have a variety of costs, Kortsarz, Krauthgamer, and Lee proved that the lowest cost vertex connectivity problem cannot be approximated to within $2^{log^{1-\epsilon}(n)}$ on any graph with n vertices [21].

2.4 Our Work

Most of our work has been on the k-ECSS problem for simple graphs. We have explored two different approximation algorithms for this problem and made improvements to each. The first algorithm is the linear programming algorithm of Gabow, Goemans, Tardos, and Williamson [12]. We improve the analysis of the algorithm in the case of simple graphs. The approximation ratio for this algorithm is based on how many edges need to be rounded. By looking at various theorems on the size of critical sets, we have improved the approximation ratio of this algorithm on undirected simple graphs from $1 + \frac{2}{k}$ to $1 + \frac{1}{k} + \frac{3}{k^2} + \frac{3}{k^2\sqrt{k}} + O(\frac{1}{k^3})$. We also improve the algorithm

by modifying the way that edges are chosen to be rounded. If we round intelligently, we can push the approximation bound even further, to $1 + \frac{1}{2k} + O(\frac{1}{k^2})$.

We have also looked at lower bounds for the linear programming algorithm. We have shown that a k-edge-connected graph with n vertices can have a laminar family with as many as $n(1 + \frac{3}{k} + \frac{1}{\sqrt{2}k\sqrt{k}} - O(\frac{1}{k^2}))$ critical sets. For the standard iterated rounding algorithm, where all edges with weights of at least $\frac{1}{2}$ are rounded up, we have an example where the algorithm gives a $1 + \frac{1}{k} + \frac{3}{k^2} + \frac{1}{\sqrt{2}k^2\sqrt{k}} - O(\frac{1}{k^3})$ approximation.

The second algorithm is Cheriyan and Thurimella's algorithm for finding a k-ECSS by using matching [1]. The approximation ratio is based on the number of "special" edges in the graph. Gabow improved the bound on the maximum number of special edges that can be in a simple graph when $k \geq 15$ [13]; we extend Gabow's bound to values $10 \leq k < 15$.

Chapter 3

Better Bounds for Approximating the Minimum k-ECSS with LP Rounding

In this chapter, we discuss our work on approximating the smallest k-edge-connected subgraph of a simple, undirected k-edge-connected graph using linear programming. Section 3.1 gives our improvements to the approximation bound. Sections 3.2 and 3.3 give lower bound examples.

Recall from Chapter 2 that linear programming approximation to the minimum k-ECSS problem works by formulating the problem as an ILP, relaxing to allow fractional edges, solving the relaxed LP, then rounding fractional edges in the LP solution. The approximation bound is based on how many edges need to be rounded and the weights on those edges.

Gabow et al. [12] proved that the linear programming technique of iterated rounding gives a bound of $1+\frac{2}{k}$. Under iterated rounding, edges with weight at least $\frac{1}{2}$ are rounded, the requirements of the linear program are updated, the linear program is solved again, and the process is repeated. Because the weights on the fractional edges are uniquely determined by a laminar family of tight sets [8], and a laminar family of subsets of an n element set can have at most $2n$ elements, there are at most $2n$ fractional edges. Rounding these will add n to the solution, which when divided by the lower bound of $\frac{kn}{2}$, gives the desired bound. Gabow et al. also gave a lower bound example to show that this bound is tight for multigraphs.

We improve the bound of Gabow et al. for simple, undirected graphs. First, we show that while an arbitrary laminar family may have $2n$ sets, a laminar family of critical sets in a simple undirected graph can have at most $n\left(1+\frac{3}{k}+\frac{3}{k\sqrt{k}}+O(\frac{1}{k^2})\right)$ sets. This immediately improves the approximation bound from $1+\frac{2}{k}$ to $1+\frac{1}{k}+\frac{3}{k^2}+\frac{3}{k^2\sqrt{k}}+O(\frac{1}{k^3})$. In addition, we give a new method

of rounding the edges that gives a bound of $1 + \frac{1}{2k} + O(\frac{1}{k^2})$ in simple, undirected graphs, a further improvement when k is large.

To demonstrate lower bounds on the approximation guarantee, we give an infinite family of examples of simple, undirected graphs which each contain a laminar family of $n\left(1 + \frac{3}{k} + \frac{1}{\sqrt{2}k\sqrt{k}} - O(\frac{1}{k^2})\right)$ critical sets. This shows that our $n\left(1 + \frac{3}{k} + \frac{3}{k\sqrt{k}} + O(\frac{1}{k^2})\right)$ bound is close to tight. We also show how to modify this infinite family of examples to give basic feasible solutions to the linear program with $n\left(1 + \frac{3}{k} + \frac{1}{\sqrt{2}k\sqrt{k}} - O(\frac{1}{k^2})\right)$ edges of weight $\frac{1}{2}$. Rounding all of these edges as is done in the traditional round-the-highest algorithm (iteratively rounding all edges with weights at least $\frac{1}{2}$) will result in an approximation ratio of $1 + \frac{1}{k} + \frac{3}{k^2} + \frac{1}{\sqrt{2}k^2\sqrt{k}} - O(\frac{1}{k^3})$. This shows that the $1 + \frac{1}{k} + \frac{3}{k^2} + \frac{3}{k^2\sqrt{k}} + O(\frac{1}{k^3})$ bound for the round-the-highest method is also close to tight.

3.1 Improved bound for LP rounding

In this section we discuss the improvements we have made to the LP rounding algorithm for finding the smallest k-edge-connected subgraph of a simple, undirected k-edge-connected graph. We will give an overview of all improvements. For full details, see Appendix A.

Recall some critical definitions. A **laminar family** $\mathcal{L}$ is a collection of sets such that for any $S, T \in \mathcal{L}$, either S and T are disjoint or one is a subset of the other. In a graph $G = (V, E)$, the degree of a vertex $v \in V$ is denoted by $d(v)$. For a subset of edges $D \subset E$, let $d_D(v)$ be the degree of v if we count only edges in D. For a real valued function x on the edges, let $d_x(v)$ be the degree of v counting each edge according to its value $x(e)$. For a subset of vertices $S \subset V$, let $d(S)$ be the number of edges that have exactly one endpoint in S, and define $d_D(S)$ and $d_x(S)$ analogously to the single vertex case.

Recall that the k-ECSS problem can be formulated as an integer linear program. For every edge e, define an indicator variable x_e and let $x_e = 1$ if we include e in our subgraph and $x_e = 0$ otherwise. Then, the k-ECSS problem reduces to the following ILP:

$$\begin{aligned} &\text{minimize} \sum_{e \in E} x_e \\ &\text{subject to :} d_x(S) \geq k && \emptyset \subset S \subset V \\ &x_e \in \{0,1\} && e \in E \end{aligned}$$

Though this ILP has an exponential number of constraints, there is an equivalent version where the number of constraints and equations is polynomial in the size of the graph [8]. A minimum solution to this ILP would give a minimum k-ECSS. As discussed in Chapter 2, the k-ECSS problem is NP hard, so obtaining an exact solution to this ILP is NP hard as well. However, we can get an approximate solution using an algorithm we call **round-the-highest**. First, we relax the conditions to allow for fractional edges:

$$\begin{aligned} &\text{minimize} \sum_{e \in E} x_e \\ &\text{subject to :} d_x(S) \geq k && \emptyset \subset S \subset V \\ &0 \leq x_e \leq 1 && e \in E \end{aligned}$$

We can solve the polynomial version of this LP in polynomial time and round some of the fractional edges, those edges e where x_e is strictly between 0 and 1 in the solution of the LP (call x_e the **weight** of edge e). We use iterated rounding to round the edges, meaning that we round only those edges that have weight at least $\frac{1}{2}$. We then update our linear program: for every edge $e \in E$ that was either rounded or had $x_e = 1$ in the solution to the current LP, add the constraint $x_e = 1$. We solve the updated LP and again round and update. We repeat this process until all constraints are satisfied. Recall from Chapter 2 that Jain [8] proved that a solution to this linear program will always have an edge of weight at least$\frac{1}{2}$. Jain also showed that the values of the fractional edges are uniquely determined by a system of linear equations, $d(S) = k$, $S \in \mathcal{L}$ where $\mathcal{L}$ is a laminar family of sets. Therefore, if OPT is the number of edges in the smallest k-ECSS, round-the-highest

will give a k-ECSS with fewer than $OPT + \frac{|\mathcal{L}|}{2}$ edges. This gives a naive approximation bound of $1 + \frac{|\mathcal{L}|}{2(OPT)} \leq 1 + \frac{|\mathcal{L}|}{kn} \leq 1 + \frac{2}{k}$ based on the fact that $OPT \geq \frac{kn}{2}$ and $|\mathcal{L}| \leq 2n$.

We make two improvements to this bound. First, while it is possible for an arbitrary laminar family of subsets of an n-element set to have $2n$ subsets, if the elements of the laminar family are critical sets of vertices in a k-edge-connected simple graph, the size of the laminar family is far more restricted. We improve the bound on the possible size of a laminar family of critical sets from $2n$ to $n\left(1 + \frac{3}{k} + \frac{3}{k\sqrt{k}} + O(\frac{1}{k^2})\right)$. Second, we suggest a more intelligent way to round the edges. Rather than rounding every edge of weight $\frac{1}{2}$ or greater, we only round one edge every iteration, giving priority to edges that we call "good" edges. This improved rounding further reduces the approximation bound.

3.1.1 Improved bound on the size of the laminar family

For any n element set, a laminar family of subsets can contain no more than $2n$ sets. If our laminar family is also a family of critical sets in a simple graph, we can improve this bound considerably.

Recall that in a k-edge-connected graph, a set S is r-critical if $V - S$ can be partitioned into exactly r sets of degree exactly k. If S is an r-critical set with $|S| = s$, then we know

$$rk \geq s(k - s + 1) \tag{3.1}$$

based on the fact that each vertex in S must have degree at least k but can have no more than $s - 1$ neighbors in S [13]. Note that a critical set is 1-critical under this notation.

Lemma 1 (Criticality)**.** *An r-critical set has cardinality $\leq r$ or $\geq k - r + 1$ if at least one of the following holds:*

(i) $r = 1$

(ii) $r = 2$ *and* $k \geq 7$

(iii) $r \leq \sqrt{k} - \frac{1}{2}$

Part (i) is a well-known result mentioned in [1] among others. Part (ii) is shown for directed graphs in [13].

Proof. The proof of all three parts follows from basic manipulation of Eq.3.1. For details, see Appendix A. □

We will divide the laminar family of critical sets into singleton sets, those with only a single vertex, and nonsingleton sets. Obviously, there can be at most n singleton sets. We will look at the family of nonsingleton sets, $\mathcal{N}$. We will use a tree notation to describe $\mathcal{N}$. A leaf of $\mathcal{N}$ will be a set $L \in \mathcal{N}$ such that there does not exist $S \in \mathcal{N}$ such that $S \subset L$. For $S, X \in \mathcal{N}$, we will say X is a descendant of S if $X \subseteq S$. We will say X is a child of S if X is a proper descendant of S and there is no $Y \in \mathcal{N}$ such that $X \subset Y \subset S$. Note that the Criticality Lemma implies that all sets of $\mathcal{N}$ must have cardinality at least k.

In order to minimize confusion, we will refer to the sets in the laminar family that make up the nodes of the tree as "sets" or "nodes" and the vertices of the original graph as "vertices."

The number of nonsingleton sets is bounded by the following theorem:

Theorem 1. *In any simple, weighted k-edge-connected graph G=(V,E) where $|V| = n$, a laminar family of nonsingleton sets of degree k has cardinality no more than $n\left(\frac{3}{k} + \frac{3}{k\sqrt{k}} + O(\frac{1}{k^2})\right)$*

Proof. Let $\mathcal{N}$ be a laminar family of nonsingleton critical sets.

We will prove the limits on the size of $\mathcal{N}$ implied by the Criticality Lemma by using a concept that we call **excess**. Excess is defined as follows: for a set $S \in \mathcal{L}$, let $X = \{X_1, \ldots, X_\ell\}$ be a collection of pairwise disjoint descendants of S. Say S has excess Δ_X over X if:

$$|S| - \sum_{i=1}^{\ell} |X_i| \geq \ell + \Delta_X$$

Note that if we let $X_1 = S$, any set will have excess -1 over itself.

The key insight is that because of criticality, any set with excess 2 actually has a larger excess if ℓ is small.

Claim 1. *If a critical set S has excess of 2 over $X = \{X_1, \ldots, X_\ell\}$ and $\ell \leq \sqrt{k} - \frac{3}{2}$, then S actually has an excess of $k - 2\ell$.*

The proof of this claim follows from part (iii) of the Criticality Lemma because $S - \bigcup X_i$ is an $(\ell+1)$-critical set (with the partition $S - \bigcup X_i, X_1, \ldots, X_\ell, V - S$) that has $\ell + 2$ vertices. This claim implies that either the excess or ℓ must be large.

To prove the $n\left(\frac{3}{k} + \frac{3}{k\sqrt{k}} + O(\frac{1}{k^2})\right)$ bound, we will give each vertex $\frac{3}{k} + \frac{3}{k\sqrt{k}} + O(\frac{1}{k^2})$ credits and show we can pay for each set using 1 of these credits. We will start with the leaves of $\mathcal{N}$ and work our way up the tree.

We will give each node of the tree a label $\Delta \in \{-1, 0, 1\}$. Leaves will be labeled as $\Delta = -1$. For a nonleaf S, let $X(S)$ be the collection of maximal proper subsets of S which all have $\Delta = -1$. If S has excess 1 over $X(S)$, we give S label $\Delta = 1$. If S has excess 0 over $X(S)$, we give S label $\Delta = 0$. If S has excess at least 2 or no more than -1 over $X(S)$, we give S label $\Delta = -1$.

We will want each set to have a certain number of credits on it based on its label. We will maintain the following invariant:

Invariant: A set S with label Δ will have $1 - \Delta$ credits remaining on it after we have paid for S and all of its descendants.

By the Criticality Lemma, leaves of $\mathcal{N}$ must have at least k vertices. Each of these k vertices has $\frac{3}{k} + \frac{3}{k\sqrt{k}} + O(\frac{1}{k^2})$ credits on it, giving a total of at least $3 + \frac{3}{\sqrt{k}} + O(\frac{1}{k})$ credits on the leaves. We can use the first 3 credits to pay for the leaf and satisfy the invariant. We will show that we can maintain the invariant for a non-leaf node S with label Δ. If S has excess less than 2 over $X(S)$, we will show that the invariant will guarantee that there will be enough credits on the children of S to pay for S and leave $1 - \Delta$ credits. If S has excess 2 over $X(S)$, then we will show that we can obtain 3 credits either from the vertices of $S - X(S)$ or from the remaining credits on the leaves.

Consider a nonleaf set S with excess less than 2 over $X(S)$. Assume that all of its descendants have been paid for and all of its children have the credits required by the invariant. First consider the case that S has only one child C, and C has label Δ_C. If $\Delta_C = -1$, then $X(S) = C$. Because

$|S - C| \geq 1$ and S does not have an excess of 2, $\Delta \geq \Delta_C + 1$. If $\Delta_C \neq -1$, then $X(S) = X(C)$. Again because $|S - C| \geq 1$ and S does not have an excess of 2, $\Delta \geq \Delta_C + 1$. In either case, there are $1 - \Delta_C \geq 2 - \Delta$ credits on C, allowing us to pay for S and maintain the invariant.

Now, consider the case that S has at least 2 children $C = \{C_1, \ldots, C_c\}$. Let Δ_i be the label of C_i. If $\Delta_i = \Delta_j = -1$ for some $i, j \leq c$ and $i \neq j$, then there are at least 4 credits on the children. This is enough to pay for S and maintain the invariant no matter what the label of S. Otherwise, if there is a child with label -1, let that child be C_1. Then $X(S) = \bigcup X(C_i) \cup \{C_1\}$ for $2 \leq i \leq c$ if $\Delta_1 = -1$ or $X(S) = \bigcup X(C_i)$ for $1 \leq i \leq c$ if $\Delta_1 \neq -1$. In either case, S has excess $\sum_{i=1}^{c} \Delta_i$ over $X(S)$. There are $c - \sum_{i=1}^{c} \Delta_i$ credits on the children of S. Under our assumptions that S has excess less than 2 over $X(S)$ and $c \geq 2$, S has label $\Delta \geq \sum_{i=1}^{c} \Delta_i$. We have at least $2 - \Delta$ credits, enough to pay for S and maintain the invariant.

This completes the proof that we can pay for all sets S with excess less than 2 and maintain the invariant, assuming that the invariant is maintained on the children. For a set C with excess 2 or greater, there are two possibilities. The first is that $|X(C)| \leq \sqrt{k} - \frac{3}{2}$, in which case, there must be at least $k - \sqrt{k} + \frac{3}{2}$ vertices in $C - X(C)$. Because each vertex has $\frac{3}{k} + \frac{3}{k\sqrt{k}} + O(\frac{1}{k^2})$ credits, there are enough credits on these vertices to pay for C and maintain the invariant. If $|X(C)| \leq \sqrt{k} - \frac{3}{2}$, we will pay for C using the credits on the leaves. Because a tree with ℓ leaves can have no more than $\frac{\ell - 1}{r - 1}$ nodes with r children, there can be no more than $\frac{\ell}{\sqrt{k} - \frac{5}{2}}$ nodes with excess 2 and $|X(C)| \geq \sqrt{k} - \frac{3}{2}$. This allows us to pay for C and maintain the invariant using the remaining $\frac{3}{\sqrt{k}} + O(\frac{1}{k})$ credits on each of the leaves. See Appendix A for the details of this case. □

This results in an immediate improvement in the approximation ratio:

Corollary 1. *Iterated rounding approximates the smallest k-ECSS to within a factor of $1 + \frac{1}{k} + \frac{3}{k^2} + \frac{3}{k^2\sqrt{k}} + O(\frac{1}{k^3})$.*

Proof. Because there is a 1-1 relationship between the fractional edges and the sets in the laminar family [8], there can be at most $n\left(1 + \frac{3}{k} + \frac{3}{k\sqrt{k}} + O(\frac{1}{k^2})\right)$ fractional edges. Because we increase the weight by at most $\frac{1}{2}$ for every fractional edge, we increase the weight of the graph by

$\frac{n}{2}\left(1+\frac{3}{k}+\frac{3}{k\sqrt{k}}+O(\frac{1}{k^2})\right)$. Dividing this by the minimum weight of an optimal k-ECSS, $\frac{kn}{2}$, gives us the desired bound. □

Both of these bounds are very close to tight. Section 3.2 gives an example of a k-edge-connected graph with a laminar family of critical sets of size

$$n\left(1+\frac{3}{k}+\frac{1}{\sqrt{2}k\sqrt{k}}-O(\frac{1}{k^2})\right).$$

Section 3.3 gives an example of an extreme point solution to the LP with

$$n\left(1+\frac{3}{k}+\frac{1}{\sqrt{2}k\sqrt{k}}-O(\frac{1}{k^2})\right)$$

edges of weight $\frac{1}{2}$. Round-the-highest gives an approximation ratio of

$$1+\frac{1}{k}+\frac{3}{k^2}+\frac{1}{\sqrt{2}k^2\sqrt{k}}-O(\frac{1}{k^3})$$

on this example.

3.1.2 Improved rounding algorithm

The approximation ratio can be improved further if, with each iteration, we only round some edges with weight at least $\frac{1}{2}$ rather than all of them. If we choose the edges to round intelligently, we can significantly improve the approximation ratio for large values of k.

Consider an extreme point solution of the LP. Let F be the set of all fractional edges. Call a fractional edge **heavy** if its weight is greater than or equal to $\frac{1}{2}$. Let $\mathcal{L}$ be the laminar family that determines the values of the fractional edges. As before, we will want to partition $\mathcal{L}$ into two subfamilies, the family of singletons $\mathcal{S}$ and the family of nonsingletons, $\mathcal{N}$. We further want to subdivide the set of singletons into:

$$\mathcal{S}_2=\{v\in\mathcal{S}: d_F(v)=2\} \qquad \mathcal{S}^3=\{v\in\mathcal{S}: d_F(v)\geq 3\}$$

At any iteration, call a heavy edge **good** if it joins two vertices of $\mathcal{S}_2$. Intuitively, a good edge is one that can be rounded cheaply because it is heavy, and rounding it will cause two singleton sets

that were not previously incident to at least k unit edges to become so. Once those two singletons are incident to k unit edges, they are no longer tight sets covering fractional edges. Thus our laminar family shrinks by two sets for the cost of rounding a single edge.

Our new algorithm is iterated rounding, with a slight modification: at every iteration we round exactly one heavy edge. If possible, we round a good edge.

Theorem 2. *The above version of iterated rounding approximates the smallest k-ECSS to within a factor of $1 + \frac{1}{2k} + O(\frac{1}{k^2})$.*

Proof. Let $\mathcal{L}$ be the laminar family corresponding to the first iteration of the algorithm where there is no good edge. F, $\mathcal{N}$, $\mathcal{S}$, etc. will all refer to this iteration.

The total number of edges that will be rounded will be the number of (good) edges rounded in previous iterations plus $|\mathcal{L}| = |\mathcal{S}| + |\mathcal{N}|$. For any good edge that was previously rounded, $e = (v, w)$, v and w will not be singletons of $\mathcal{L}$. Thus, there can be at most $\frac{n-|\mathcal{S}|}{2}$ edges rounded in previous iterations. Rounding these will add at most $\frac{n-|\mathcal{S}|}{4}$ to the weight of the solution. The value of $|\mathcal{N}|$ is bounded by Theorem 1. We can bound $|\mathcal{S}|$ by bounding $|\mathcal{S}_2|$ and $|\mathcal{S}^3|$. We bound $|\mathcal{S}^3|$ using the handshaking lemma:

$$2|\mathcal{S}_2| + 3|S^3| \leq \sum_{v \in V} d_F(v) = 2(|\mathcal{S}| + |\mathcal{N}|) = 2(|\mathcal{S}_2| + |\mathcal{S}^3| + |\mathcal{N}|)$$

This simplifies to $|\mathcal{S}^3| \leq 2|\mathcal{N}|$.

Note that because the weights of the fractional edges incident to a vertex of $\mathcal{S}_2$ must sum to 1 (because a singleton set must have degree exactly k). Thus, every vertex in $\mathcal{S}_2$ must be incident to a heavy edge. If the other endpoint of that heavy edge was a vertex in S_2, then it would be a good edge, contradicting our assumption that there is no good edge at this iteration of the algorithm. Therefore, every vertex in $\mathcal{S}_2$ must have a fractional edge leading to a vertex not in $\mathcal{S}_2$. This allows us to bound $|\mathcal{S}_2|$:

$$|\mathcal{S}_2| \leq \sum_{v \notin \mathcal{S}_2} d_F(v) = 2(|\mathcal{S}| + |\mathcal{N}|) - 2\mathcal{S}_2 = 2(|\mathcal{S}^3| + |\mathcal{N}|) \leq 6|\mathcal{N}|$$

So $|\mathcal{S}| = |\mathcal{S}_2| + |\mathcal{S}^3| \leq 8|\mathcal{N}|$.

Overall, we will increase the objective function by no more than $\frac{n-|\mathcal{S}|}{4} + \frac{|\mathcal{L}|}{2} = \frac{n+|\mathcal{S}|}{4} + \frac{|\mathcal{N}|}{2} \leq \frac{n}{4} + \frac{5|\mathcal{N}|}{2}$. By Theorem 1, $|\mathcal{N}| = O(\frac{1}{k})$, so substituting in this value and dividing by $\frac{kn}{2}$ gives the desired approximation ratio. □

This analysis can be refined further to improve the constant factor on the $\frac{1}{k^2}$ term. The details are in Appendix A, Section 4.

3.2 Laminar family lower bound example

3.2.1 Overview

We now exhibit an infinite family of graphs $G = (V, E)$ with a large laminar family of critical sets. The size of this laminar family is close to the upper bound proved in Section 3.1.1. Specifically, we will prove the following theorem:

Theorem 3. *For values of $k \geq 8$ that are twice the value of an even perfect square, there exists a graph $G = (V, E)$ with a laminar family of critical sets of size*

$$n\left(1 + \frac{3}{k} + \frac{1}{\sqrt{2}k\sqrt{k}} - O(\frac{1}{k^2})\right).$$

We prove this theorem by constructing the graph. We first give an overview of the construction before moving into the details in Section 3.2.2. In Section 3.2.3 we prove the example is k-edge-connected and calculate the number of sets in the laminar family.

Our laminar family will consist of several types of sets as illustrated in Fig. 3.1. All vertices will be singleton sets. An **L-set** will be a critical set of $k+1$ vertices containing a critical subset of k vertices. We will have two types of **S-sets**. A minimal S-set will be a critical set of $k+2$ vertices that contains a critical subset of $k+1$ vertices, which in turn contains a critical subset of k vertices. Larger S-sets will be the union of an S-set and an L-set. We will say that an S-set is at **height** i if there are i smaller S-sets contained in it, so an S-set at height i, $i > 0$, will be the union of an L-set and an S-set at height $i-1$. Finally, **T-sets** will be the union of two S-sets. We

define

$$h = \frac{k}{2} \tag{3.2}$$

and

$$\sigma = \sqrt{h}. \tag{3.3}$$

In the terminology used in the upper bound proof in Section 2, an L-set is a small chain node whose only child is a leaf. A minimal S-set is a small chain node whose child is another small chain node and whose grandchild is a leaf, while a larger S-set is a branching node with two children. A T-set is also branching node with two children. These sets are differentiated by their excess. An L-set has an excess of 0 over its (leaf) child. An S-set has an excess of 1 over $\{X : X \subset S$ and X is a leaf $\}$. A T-set has an excess of 2 over $\{X : X \subset T$ and X is a leaf $\}$. The T-set contains $2\sigma = \sqrt{2k} > \sqrt{k} - 3/2$ leaves, so it does not violate Claim 1 of Theorem 2.2.

3.2.2 The Construction

L-sets are referred to as L_i, $1 \leq i \leq \sigma - 1$, where i is the height of the smallest S-set containing the L-set. L_i has $k+1$ vertices, $\{0, 1, \ldots, k-1, \ell_i\}$. Let $A_i = \{0, 1, \ldots, k-1\}$. We construct edges incident to the vertices of L_i so that the following conditions hold: $d(\ell_i, A_i) = d(\ell_i, \overline{L}_i) = h$ and $d(A_i) = d(L_i) = k$. The vertices of A_i form a complete graph. Also, for h vertices $u \in A_i$ there is an edge (u, ℓ_i) and $d(u, \overline{L}_i) = 0$; for the remaining h vertices, there is no edge (u, ℓ_i) and $d(u, \overline{L}_i) = 1$.

Minimal S-sets are referred to as S_0. S_0 has $k+2$ vertices, $\{0, 1, \ldots, k-1, s, t\}$, where we let $A_0 = \{0, 1, \ldots, k-1\}$ and $B_0 = A_0 \cup \{s\}$. Notice that sets A_i refer to the grandchild of S_0 when $i = 0$ and to the child of an L-set when $i > 0$. We construct edges incident to the vertices of S_0 so the following conditions hold: $d(s, A_0) = d(t, A_0) = d(s, \overline{S}_0) = d(t, \overline{S}_0) = h$ and $d(A_0) = d(B_0) = d(S_0) = k$. The vertices of A_0 form a complete graph. For h vertices $u \in A_0$ there is an edge (u, s), and for the remaining h vertices there is an edge (u, t). For all $u \in A_0$, $d(u, \overline{S}_0) = 0$.

Larger S-sets are defined inductively. For an S-set at height r, $1 \leq r \leq \sigma - 1$, $S_r =$

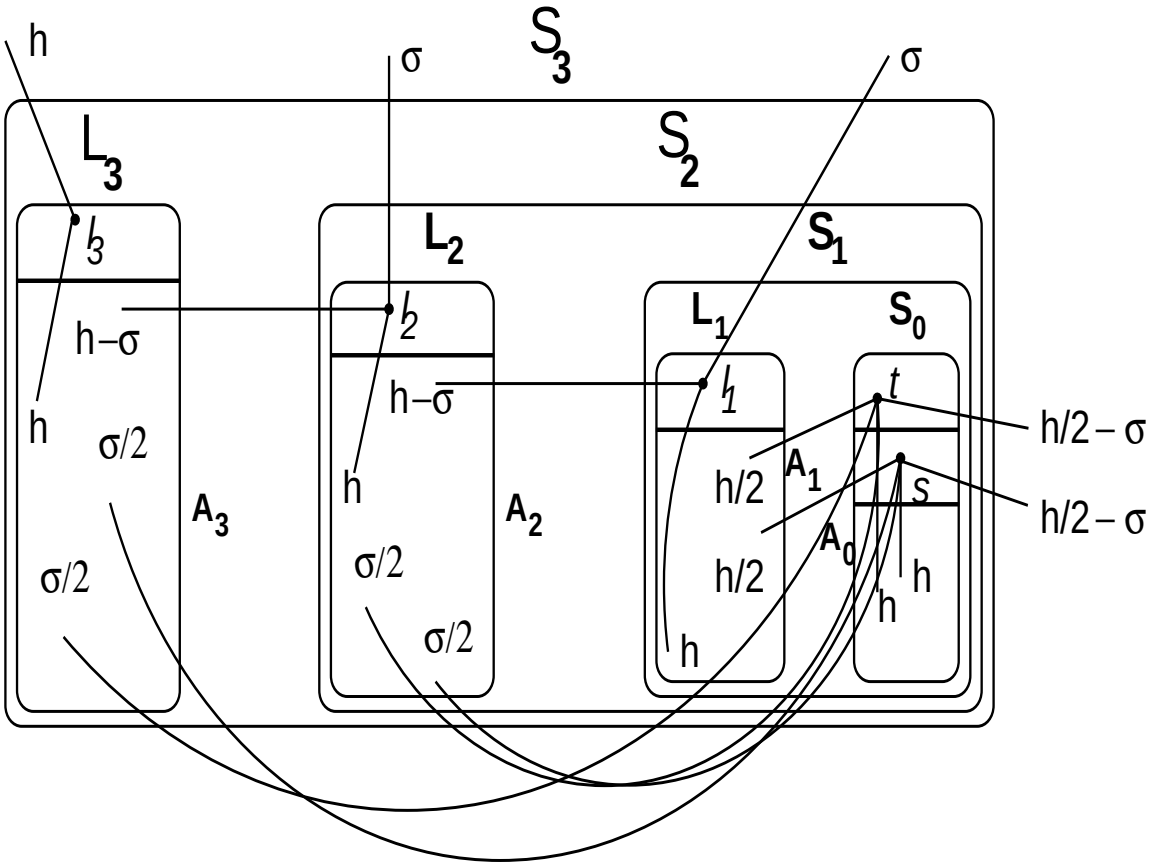


Figure 3.1: An S-set at height 3 and all its descendants.

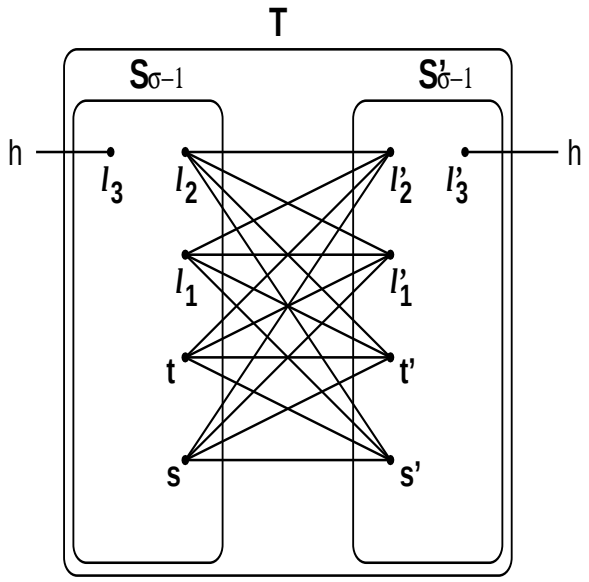


Figure 3.2: The edges between the two $S_{\sigma-1}$ sets in a T-set that has $\sigma = 4$.

$S_{r-1} \cup L_r$. Observe that since S_r contains only one S_0, the vertices $s, t \in S_r$ are unique. Given $d(S_{r-1}) = d(L_r) = k$, we will arrange the edges so that $d(S_{r-1}, L_r) = h$ as well as $d(S_r) = k$. The edges are arranged as follows: if $r > 1$, then $d(A_r, s) = d(A_r, t) = \sigma/2$ and $d(A_r, \ell_{r-1}) = h - \sigma$; otherwise, if $r = 1$, $d(A_1, s) = d(A_1, t) = h/2$. For the edges leaving S_r, let $d(\ell_r, \overline{S}_r) = h$, $d(\{s, t\}, \overline{S}_r) = h - (r-1)\sigma$, and $d(\ell_i, \overline{S}_r) = \sigma$ for all $1 \leq i \leq r-1$.

A T-set consists of two S-sets at height $\sigma - 1$, $S_{\sigma-1}$ and $S'_{\sigma-1}$. There are h edges between the two S-sets and k edges leaving the T-set. Note $d(\ell_i, \overline{S}_{\sigma-1}) = d(\ell'_i, \overline{S'}_{\sigma-1}) = \sigma$ for $1 \leq i \leq \sigma - 2$; also, $d(s, \overline{S}_{\sigma-1}) = d(t, \overline{S}_{\sigma-1}) = d(s', \overline{S'}_{\sigma-1}) = d(t', \overline{S'}_{\sigma-1}) = h/2 - \sigma(\sigma-2)/2 = h/2 - \sigma^2/2 + \sigma = \sigma$. Thus, we can form a complete bipartite graph using s, t, ℓ_i, $1 \leq i \leq \sigma - 2$ and their counterparts in $S'_{\sigma-1}$, as in Fig. 3.2. For the edges leaving the T-set, $d(\ell_{\sigma-1}) = d(\ell'_{\sigma-1}) = h$. This gives us $2h = k$ edges leaving the T-set as well as $d(S_{\sigma-1}, S'_{\sigma-1}) = \sigma^2 = h$ as desired.

To complete the example, take a prime number of T-sets $p > h$. Label the T-sets as $T_0, T_1, ..., T_{p-1}$. Connect the T-sets using h different cycles. There are two cycles of the form $T_0, T_1, T_2, \ldots, T_{p-1}, T_0$, one cycle through the $\ell_{\sigma-1}$ of each T-set and one cycle through the $\ell'_{\sigma-1}$. Similarly, there are two cycles of the form $T_0, T_2, T_4, \ldots, T_{p-1}$, $T_1, \ldots, T_{p-2}, T_0$, two cycles of the

form $T_0, T_3, T_6, \ldots, T_{p-3}, T_0$, up through $T_0, T_{h/2}, T_h, \ldots, T_0$. A prime number of T-sets is used in order to ensure that all of these cycles contain all T-sets.

3.2.3 Analysis

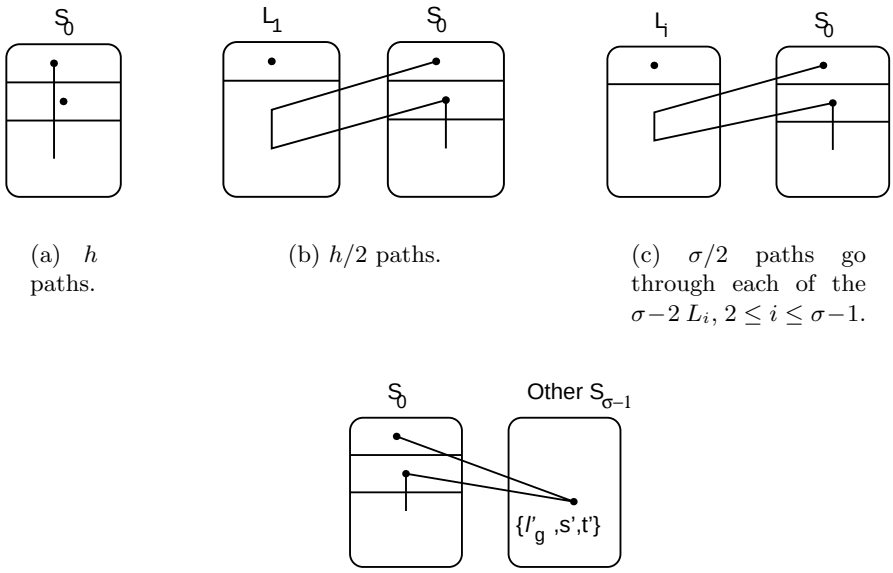
In order to prove that the example graph is k-edge-connected, we first show (in Fig. 3.3) that for any $v \in B_0$, there are k edge-disjoint paths from v to t. This allows us to contract S_0 into a single vertex.

Next we prove that from any ℓ_i, $1 \leq i \leq \sigma - 2$, there are k edge-disjoint paths to ℓ_{i+1} (Fig. 3.4). This allows us to contract all ℓ_i, $1 \leq i \leq \sigma - 1$, into a single vertex, ℓ. The following facts are used to validate Fig. 3.4: in Fig. 3.4(a) and (d), there are h paths that go through A_{i+1} to ℓ_{i+1}, which each use exactly one of the h edges between A_{i+1} and ℓ_{i+1}. For any $j > 1$, the $h - \sigma$ edges from L_j to ℓ_{j-1} occur $\leq \sigma(\sigma - 1) = h - \sigma$ total times in the paths of Fig. 3.4(b), (e), and (f). Similarly, in Fig. 3.4(b), (d), and (e), in the case $i = 1$, the h edges from A_1 to S_0 are each used exactly once. Also, for Fig. 3.4(e) and (f), G has $h - \sigma$ edge-disjoint paths from $\{s', t', \ell'_g : 1 \leq g \leq \sigma - 2\}$ to $\ell'_{\sigma-1}$. This can be increased to h edge-disjoint paths if we include the σ edge-disjoint paths from $\{s', t'\}$ through $A'_{\sigma-1}$ to $\ell'_{\sigma-1}$, which allows us to have h paths going through the other T-sets in the case $i = \sigma - 2$. If $i = \sigma - 2$, the paths in Fig. 3.4(c) go to $S'_{\sigma-1}$ to $\ell'_{\sigma-1}$ and into another T-set. Also in Fig. 3.4(c),(e), and (f), a path through another T-set starts at that T-set's $\ell'_{\sigma-1}$, goes through the T-set to $\ell_{\sigma-1}$, and from there to the $\ell_{\sigma-1}$ of our initial T-set. Because the T-sets are connected and each of these paths goes through a different T-set, the details of the path through the T-set are irrelevant. Note that there are potentially $\sigma(\sigma - 3) + 2\sigma + \sigma = h$ paths of this type. Because each T-set has two edges to h other T-sets, one from $\ell_{\sigma-1}$ to the other T-set's $\ell_{\sigma-1}$ and one from $\ell'_{\sigma-1}$ to the other T-set's $\ell'_{\sigma-1}$, each of these paths can go through a different T-set.

Next we prove that there are k edge-disjoint paths between S_0 and ℓ (Fig. 3.5). For the 2σ paths in Fig. 3.5(b), we use the inequality $h \geq 2\sigma$, which holds for $k \geq 8$. This allows us to contract S_0 and ℓ. For any vertex $v \in A_i$, $1 \leq i \leq \sigma - 1$, it is obvious that there are k edge-disjoint paths

to $\ell \cup S_0$ because all k edges leaving A_i go to either ℓ or S_0. This allows us to contract $S_{\sigma-1}$ into a single vertex.

Between any two $S_{\sigma-1}$ in the same T-set, there are h paths from the top-level cycles, and h paths within the T-set, giving us k edge-disjoint paths between them, and allowing us to contract each T-set into a single vertex. The T-sets are k-edge-connected because there are h cycles through them.



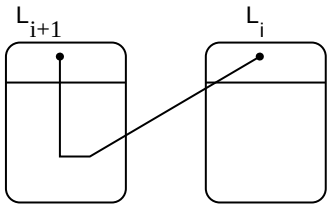
(a) h paths.

(b) $h/2$ paths.

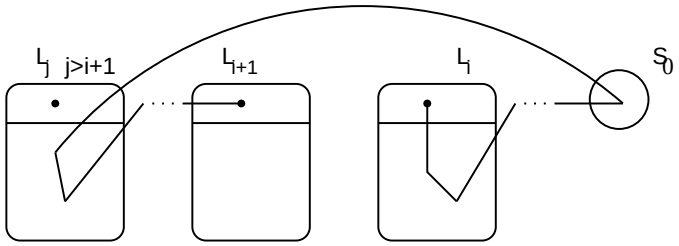
(c) $\sigma/2$ paths go through each of the $\sigma-2$ L_i, $2 \leq i \leq \sigma-1$.

(d) σ paths go through the other $S_{\sigma-1}$ in the same T-set.

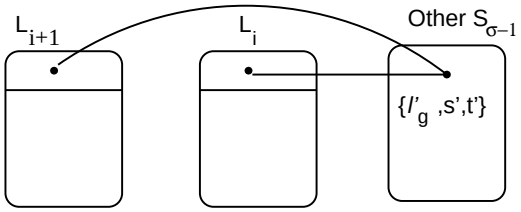
Figure 3.3: $h + h/2 + (\sigma^2 - 2\sigma)/2 + \sigma = k$ paths from a vertex in A_0 to t. These same paths can also be used to give k edge-disjoint paths from s to t, although the lengths of the paths will be slightly different from what is pictured here.



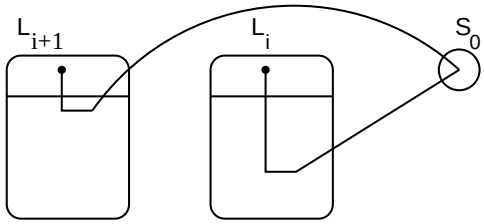
(a) $h - \sigma$ paths.



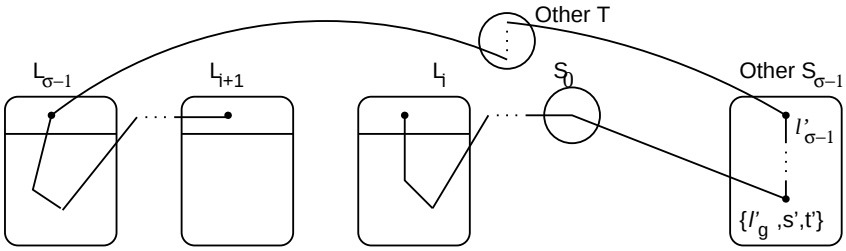
(b) σ paths through each L_j, $j > i + 1$, for a total of $\sigma(\sigma - 1 - (i + 1))$.



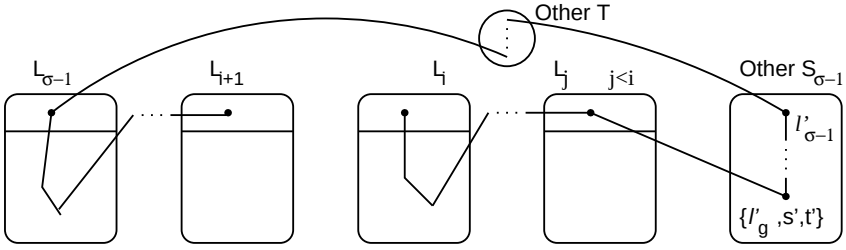
(c) σ paths (if $i = \sigma - 2$, these paths go through the other T-sets).



(d) σ paths.

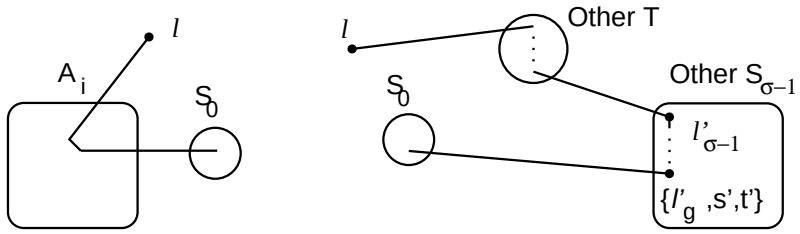


(e) 2σ paths.



(f) σ paths through each L_j, $j < i$, for a total of $\sigma(i - 1)$.

Figure 3.4: $(h-\sigma)+\sigma(\sigma-i-2)+\sigma+\sigma+2\sigma+\sigma(i-1) = k$ paths from ℓ_i to ℓ_{i+1} for $1 \leq i \leq \sigma - 2$.



(a) $h+\sigma(\sigma-2) = k-2\sigma$ paths.

(b) 2σ paths.

Figure 3.5: $(k-2\sigma)+2\sigma = k$ paths from S_0 to ℓ.

We now calculate the size of $\mathcal{L}$ in terms of the number of vertices in the graph, n. We do this in two stages: first, we calculate the number of vertices contained in each T-set, n_T. Next we calculate a, the total number of non-singleton sets that each T-set contributes to $\mathcal{L}$. Then, the size of $\mathcal{L}$ is the number of T sets multiplied by the number of sets each contributes to $\mathcal{L}$ plus the n singletons, $n(1+a/n_T)$.

Each T-set contains two S-sets at height $\sigma-1$. This means each T-set contains $2\sigma-2$ L-sets, which each contain $k+1$ vertices. Each T-set also contains 2 S_0, which each contain $k+2$ vertices. Thus $n_T = 2\sigma(k+1)+2$. Now, each of the $2\sigma-2$ L-sets contributes 2 non-singleton sets. Each of the 2 S_0 contributes 3. There are $2\sigma-2$ larger S-sets, as well as the T-set itself. Therefore $a = 2(2\sigma-2)+6+(2\sigma-2)+1 = 6\sigma+1$. This gives us a total size for $\mathcal{L}$ of $n\left(1+\frac{6\sigma+1}{2\sigma(k+1)+2}) = n(1+\frac{3}{k}+\frac{1}{2\sigma k}-O(1/k^2)\right)$. When we substitute for $\sigma = \sqrt{h} = \sqrt{\frac{k}{2}}$, we get a laminar family of size $n\left(1+\frac{3}{k}+\frac{1}{\sqrt{2}k\sqrt{k}}-O(\frac{1}{k^2})\right)$.

3.3 Round-the-highest lower bound example

3.3.1 Overview

In the previous section, we gave an example of a k-edge-connected graph G with n vertices that has a laminar family which contains $n\left(1+\frac{3}{k}+\frac{1}{\sqrt{2}k\sqrt{k}}-O(\frac{1}{k^2})\right)$ critical sets. In this section, we will show how to modify that example to give an infinite family of graphs where the LP from

Section 3.1 has a basic feasible solution (bfs) with a large number of fractional edges. We will prove the following theorem:

Theorem 4. *For values of $k \geq 8$ that are twice the value of an even perfect square, there exists a graph $G = (V, E)$ where a solution to the linear program in Section 3.1 has $n\left(1 + \frac{3}{k} + \frac{1}{\sqrt{2}k\sqrt{k}} - O(\frac{1}{k^2})\right)$ fractional edges, all of which have weight $\frac{1}{2}$. Rounding all of these edges as is done in the traditional round-the-highest method will result in an approximation ratio of*

$$1 + \frac{1}{k} + \frac{3}{k^2} + \frac{1}{\sqrt{2}k^2\sqrt{k}} - O(\frac{1}{k^3}).$$

As in the previous section, we will first give a brief overview of the construction before going into the details in Section 3.3.2. Section 3.3.3 will prove that the solution to the LP that we give is in fact a bfs. Section 3.3.4 gives a count of the fractional edges.

For this construction, we will need a few additional definitions. Call an edge of weight 1 a **unit edge** and an edge with weight w, $0 < w < 1$, a **fractional edge**. For any real-valued variables x and y, we say x and y are **complementary** if $x + y = 1$. We denote this by $x = \overline{y}$. For a vertex v, let $d_U(v)$ be the number of unit edges incident to v and $d_F(v)$ be the number of fractional edges incident to v. These will also be referred to as the **unit degree** and **fractional degree** respectively. For sets $S, T \subset V$, define $d_U(S)$, $d_F(S)$, $d_U(S, T)$, $d_F(S, T)$ analogously.

We will describe G by describing the bfs and the laminar family $\mathcal{L}$ of tight sets associated with the bfs. We will focus on the differences between this construction and the previous one. As in the previous example, every vertex of G will be a singleton set. The nonsingleton sets will be classified as either L-, S-, or T-sets. As before, all S-sets and L-sets will have a height i: S_i will contain i smaller S-sets within it and will be the smallest S-set that contains L_i. A T-set will consist of two S-sets, one at height σ and one at height $\sigma - 1$. Note that the T-set is different from the previous example, where a T-set consisted of two S-sets at height $\sigma - 1$.

As we construct each set, we will be mindful of the fractional edges within that set and leaving the set. We will maintain the following invariants:

I1: For any L-, S- or T-set, any two fractional edges within the set will either have identical or complementary weights.

I2: S_i will have two pairs of fractional edges with complementary weights leaving it if i is even or three pairs if i is odd.

We will use these invariants to prove that the weights of the fractional edges are unique, one of the conditions necessary for this solution to be a bfs.

3.3.2 The Construction

In this construction, L_i will consist of $k+2$ vertices, $\{0, 1, \ldots, k, \ell_i\}$. Let $A_i = \{0, 1, \ldots, k\}$. There will be an edge between every two vertices of A_i, but those shown in Fig. 3.6 will be fractional edges. There will be two fractional edges and $h-1$ unit edges from A_i to ℓ_i. There will be two fractional edges and $h-1$ unit edges leaving A_i. For all values of $i < \sigma$, there will be four fractional edges and $h-2$ unit edges from ℓ_i leaving L_i. For $i = \sigma$, there will be six fractional and $h-3$ unit edges leaving ℓ_i. See Fig. 3.6 for details. Note that the labels b_1, b_2, c, d, e, and f on the fractional edges leaving L_i are temporary labels used in this example to determine the relative values of these fractional edges. In later stages of the construction, the labels on these edges will differ depending on the exact value of i.

For the fractional edges of L_i, $1 \leq i \leq \sigma - 1$, shown in Fig. 3.6, the following must be true (notation defined in Fig. 3.6):

$$2a + b_1 + b_2 = 2 \text{ because } d(A_i) = k$$

$$b_1 + b_2 + c + d + e + f = 3 \text{ because } d(L_i) = k$$

$$2a + c + d + e + f = 3 \text{ because } d(\ell_i) = k$$

Combining the second and third of these gives us $2a = b_1 + b_2$. This can be combined with the first equation to prove $a = \frac{1}{2}$, $b_1 = \overline{b}_2$, and $c + d + e + f = 2$. Also, the fact that $b_1 + b_2 = 1$

implies that all edges in the odd cycle $0, 1, \ldots, k-3, k-2, 0$ shown in Fig. 3.6 must have weight $\frac{1}{2}$. This proves Invariant I1. A similar argument can be made for L_σ except that there are six edges from ℓ_σ that leave L_σ, and their weights will be forced to add up to 6.

S_0 will have $k+3$ vertices, $\{0, 1, \ldots, k, s, t\}$. $A_0 = \{0, 1, \ldots, k\}$ and $B_0 = A_0 \cup \{s\}$. Again, there will be an edge between every two vertices of A_0, but those shown in Fig. 3.7 will be fractional edges. We arrange the four fractional edges leaving A_0 as follows: $d_F(A_0, s) = 2$, $d_F(A_0, t) = 1$, $d_F(A_0, \overline{S}_0) = 1$. There is a fractional edge between s and t. There is one fractional edge from s and two from t that leave S_0. For the unit edges, we have

$$d_U(A_0, s) = d_U(A_0, t) = d_U(s, \overline{S}_0) = d_U(t, \overline{S}_0) = h - 1$$

In A_0, vertices $k-1$ and k each have $k-1$ unit edges to other vertices in A_0, one fractional edge to a vertex of A_0, and one fractional edge to $\{s, t\}$. The fractional edges must be complementary to force these vertices to have degree k. All other vertices in A_0 have $k-2$ unit edges and two fractional edges to vertices in A_0. Vertex 0 has two fractional edges leaving S_0. All vertices except for 0, $k-1$, and k have 1 unit edge leaving S_0, so each of these vertices has a total of $k-1$ unit edges. Again, this forces the fractional edges adjacent to each of these vertices to be complementary. There are a total of $k-2$ unit edges that leave S_0. This is the right number for there to be $h-1$ edges to s and $h-1$ to t as specified in the description of S_0.

The following must be true about the value of the fractional edges of S_0:

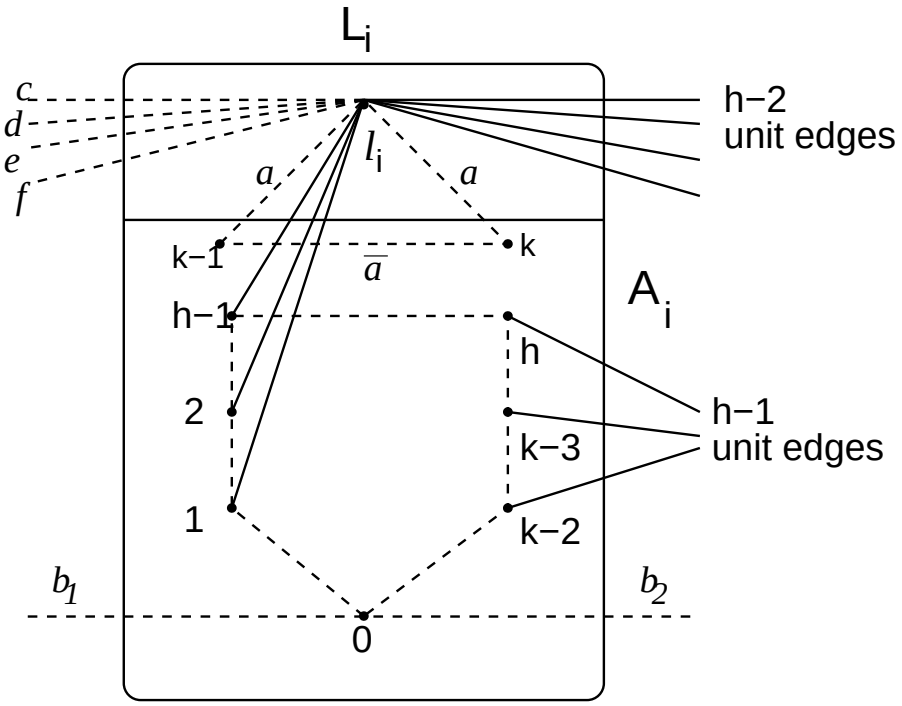


Figure 3.6: Fractional edges of L_i, $1 \leq i \leq \sigma - 1$. L_σ is the same except $d_F(\ell_\sigma, \overline{L}_\sigma) = 6$ and $d_U(\ell_\sigma, \overline{L}_\sigma) = h - 3$.

$$\overline{a}_1 + a_2 + \overline{a}_3 + a_4 = 2 \text{ because } d(A_0) = k$$

$$\overline{a}_3 + a_4 + b_1 + b_2 = 2 \text{ because } d(B_0) = k$$

$$a_4 + b_1 + c_1 + c_2 = 2 \text{ because } d(S_0) = k$$

$$a_1 + a_2 + a_3 + a_4 = 2 \text{ because } d(0) = k$$

$$\overline{a}_1 + a_2 + b_1 + b_2 = 2 \text{ because } d(s) = k$$

$$\overline{a}_3 + b_2 + c_1 + c_2 = 2 \text{ because } d(t) = k$$

Combining the equations for B_0, s, and A_0 gives us $a_1 = a_2$, $a_3 = a_4$, and $b_1 = \overline{b}_2$. Combining these with the equation for v_A gives $a_1 = a_2 = \overline{a}_3 = \overline{a}_4$. Finally if we add in the equations for S_0 and t, we can show $b_1 + a_4 = 1$ which implies $a_1 = a_2 = \overline{a}_3 = \overline{a}_4 = b_1 = \overline{b}_2$ and $c_1 + c_2 = 1$. This proves Invariants I1 and I2.

Just as in Section 3.2, for $0 < i \leq \sigma - 1$, $S_i = S_{i-1} \cup L_i$. There are slightly fewer unit edges than there were in the previous example, but the arrangement of edges is almost the same. The arrangement of unit edges is shown in Fig. 3.8. The following changes need to be made for edges within S_i: $d_U(A_1, s) = \frac{h}{2} - 1$ and $d_U(A_i, \ell_{i-1}) = h - 2 - \sigma$ for $i > 1$. A few additional changes need to be made for edges leaving S_i: $d_U(\ell_i, \overline{S}_i) = h - 2$, $d_U(t, \overline{S}_i) = \frac{h}{2} - 1 - \frac{(i-1)\sigma}{2}$. In addition, $d_U(A_{2j}, A_{2j+1}) = 1$ for values of $j \geq 1$. All other unit edges are the same as in the previous example: $d_U(A_1, t) = \frac{h}{2}$, $d_U(A_i, s) = d_U(A_i, t) = \frac{\sigma}{2}$ for $i > 1$, $d_U(s, \overline{S}_i) = \frac{h}{2} - \frac{(i-1)\sigma}{2}$, and $d_U(\ell_j, \overline{S}_i) = \sigma$ for all $1 \leq j \leq i - 1$.

The arrangement of the fractional edges of S_i is slightly different depending on whether i is even or odd. Both arrangements are shown in Fig. 3.9. For both even and odd values of i, $d_F(\ell_i, \ell_{i-1}) = d_F(\ell_{i-1}, \overline{S}_i) = d_F(\ell_{i-2}, \overline{S}_i) = 1$ (for purposes of this discussion, let $\ell_0 = t$ and $\ell_{-1} = s$). In addition, $d_F(\ell_i, \overline{S}_i) = 2$. For odd values of i, $d_F(A_i, \overline{S}_i) = 2$ and $d_F(\ell_i, \ell_{i-3}) = 1$ (if $i = 1$, this edge is from ℓ_1 to A_0). For even values of i, $d_F(A_i, A_{i-1}) = d_F(\ell_i, A_{i-1}) = d_F(A_i, \ell_{i-3}) = 1$.

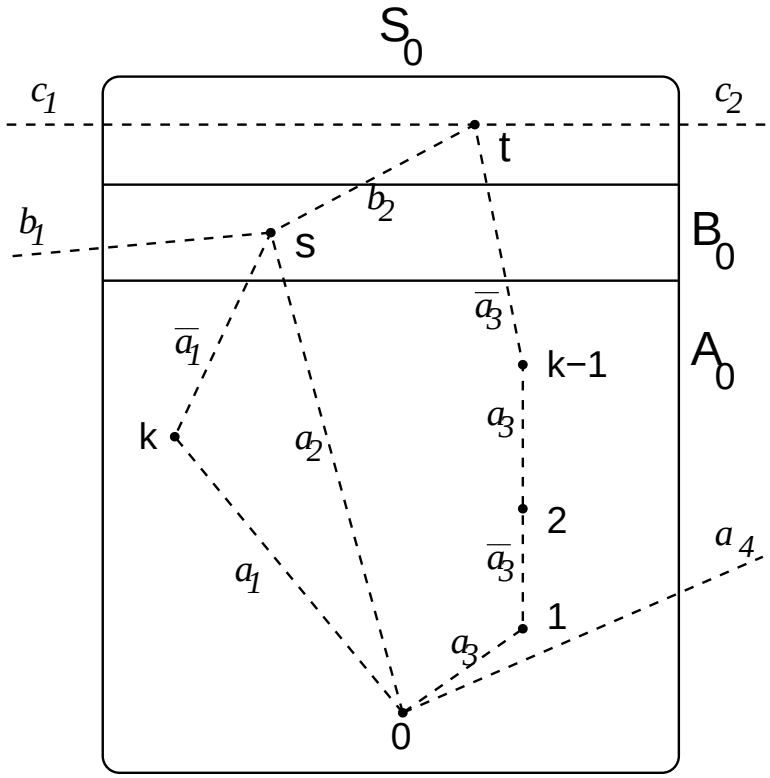


Figure 3.7: Fractional edges of S_0.

A slightly different proof is needed to prove the invariants on the fractional edges depending on whether i is even or odd. For S_1, the following is true about the fractional edges:

$$c_1 + c_2 = e_1 + e_2 = b_1 + a_4 = 1 \text{ from prior constraints}$$
$$a_4 + c_1 + d_1 + d_2 = 2 \text{ because } d(\ell_1) = k$$
$$b_1 + c_2 + d_1 + d_2 = 2 \text{ because } d(S_1) = k$$

These last two equations can be combined to give $a_4 + c_1 = b_1 + c_2$. Combining these with information from the previous constraints gives $c_1 + \overline{b}_1 = b_1 + \overline{c}_1$. This implies $c_1 = b_1$ and $a_4 = \overline{b}_1 = \overline{c}_1 = c_2$. This also implies $d_1 + d_2 = 1$. These equations prove Invariants I1 and I2.

Note that while these specific equations refer to S_1, the invariants of this construction mean that identical equations will hold for all S_i where i is odd.

For even values of i, note the following about S_2:

$$b_1 + c_2 = d_1 + d_2 = e_1 + e_2 = b_1 + e_2 = 1 \text{ from prior constraints}$$
$$d_2 + e_1 + f_1 + f_2 = 2 \text{ because } d(\ell_2) = k$$
$$c_2 + d_1 + f_1 + f_2 = 2 \text{ because } d(S_2) = k$$

Adding those last two equations gives us

$$d_1 + e_1 + c_2 + d_2 + 2(f_1 + f_2) = (d_1 + d_2) + (c_2 + e_1) + 2(f_1 + f_2) = 4$$

From the equations from the previous sets, we know $d_1 + d_2 = 1$, and we can deduce $c_2 + e_1 = 1$. This gives us $f_1 + f_2 = 1$ and also

$$b_1 = \overline{c}_2 = d_1 = \overline{d}_2 = \overline{e}_1 = e_2$$

proving Invariants I1 and I2.

Again, while these specific equations refer to S_2, the invariants mean that we have identical equations for all S_i where i is an even number less than σ.

When $i = \sigma$, there are a few differences. Neither L_σ nor S_σ are exactly the same as L_i or S_i for smaller values of i. At height σ, $d_F(\ell_\sigma, \overline{L}_\sigma) = 6$, $d_F(\ell_\sigma, \overline{S}_\sigma) = 4$, and $d_U(\ell_\sigma, \overline{L}_\sigma) = d_U(\ell_\sigma, \overline{S}_\sigma) = h - 3$. The fractional edges are arranged as in Fig. 3.10. The unit edges are as shown in Fig. 3.11.

The values of the fractional edges in S_σ are almost identical to those at other even heights:

$$x_1 + x_2 = t_1 + t_2 = w_1 + w_2 = x_1 + w_2 = 1 \text{ from prior constraints}$$

$$t_2 + w_1 + y_1 + y_2 + z_1 + z_2 = 3 \text{ because } d(\ell_\sigma) = k$$

$$x_2 + t_1 + y_1 + y_2 + z_1 + z_2 = 3 \text{ because } d(S_\sigma) = k$$

Just as in the other even cases, we can add those last two equations to get

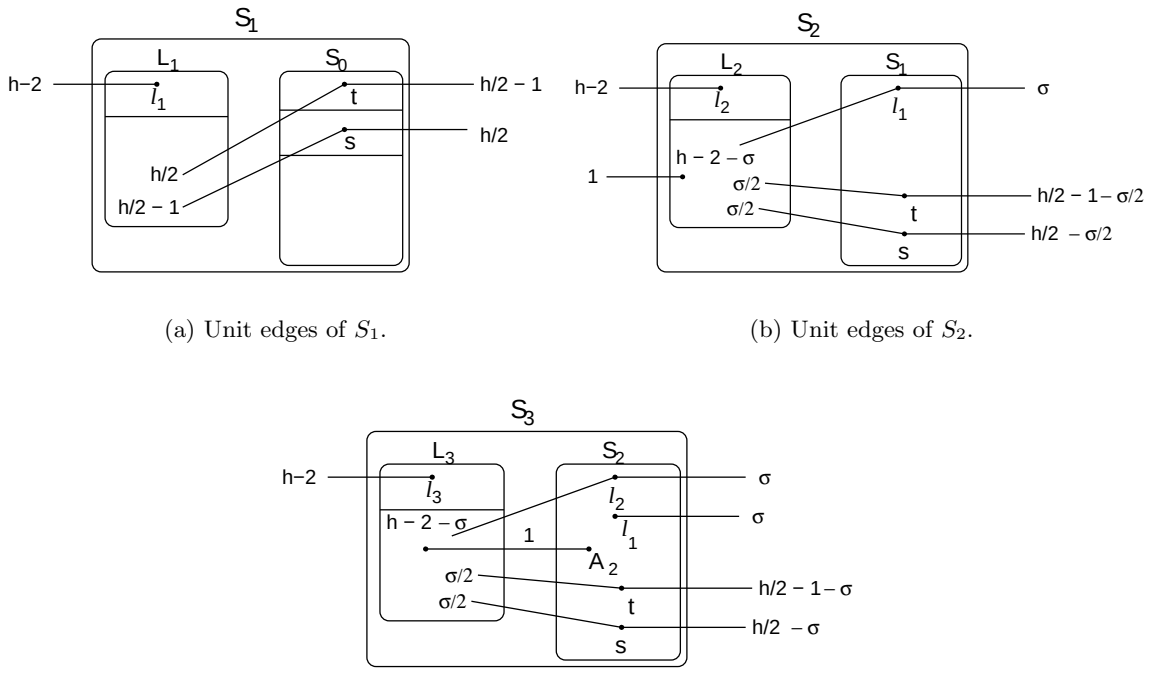
$$(t_1 + t_2) + (x_2 + w_1) + 2(y_1 + y_2 + z_1 + z_2) = 6$$

From the equations from the previous sets, we know $t_1 + t_2 = 1$. From the equations above derived from prior constraints, we can deduce that $x_2 + w_1 = 1$. This gives us $y_1 + y_2 + z_1 + z_2 = 2$ and also

$$x_1 = \overline{x}_2 = t_1 = \overline{t}_2 = w_1 = \overline{w}_2$$

which proves Invariant I1 for S_σ.

A T set is the union of two S-sets at heights σ and $\sigma - 1$, $T = S_\sigma \cup S'_{\sigma-1}$. The fractional edges are arranged as in Fig. 3.12. The unit edges are shown in Fig. 3.13. For all values $1 \leq i \leq \sigma - 1$ and $1 \leq j \leq \sigma - 2$ there is a unit edge between ℓ_i and ℓ'_j. There are also unit edges between ℓ_i and s' as well as ℓ_i and t'. If j is odd, there will be a unit edge between ℓ'_j and s. If j is even, there will be a unit edge between ℓ'_j and t. For unit edges leaving T, we have $d_U(\ell_\sigma, \overline{T}) = h - 3$, $d_U(\ell'_{\sigma-1}, \overline{T}) = h - 2$, and $d_U(s, \overline{T}) = d_U(s', \overline{T}) = d_U(A_\sigma, \overline{T}) = 1$.

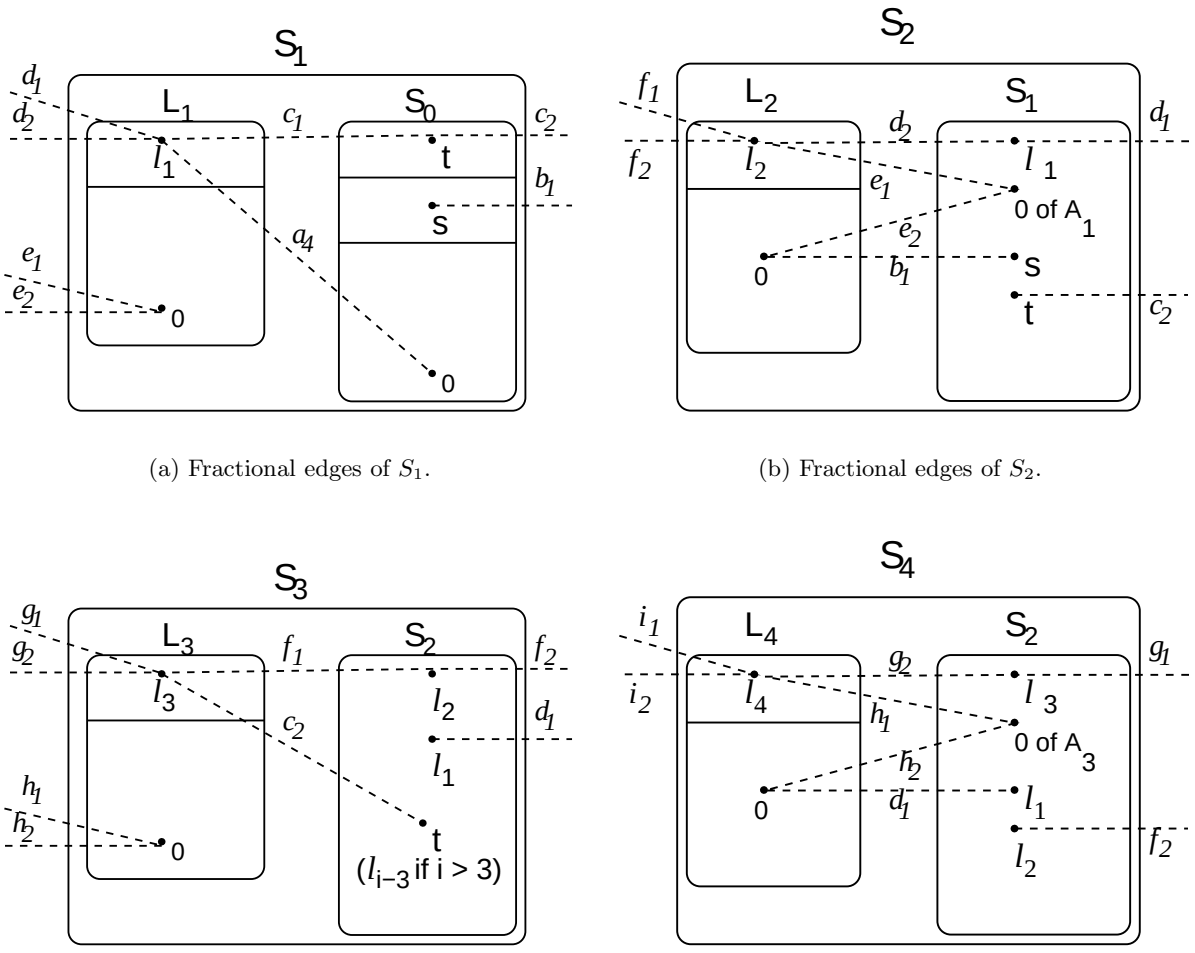


(a) Unit edges of S_1.

(b) Unit edges of S_2.

(c) Unit edges of S_3.

Figure 3.8: Unit edges of S_1, S_2, and S_3. For larger values of i, the arrangement will be similar to S_2 if i is even or S_3 if i is odd.



(a) Fractional edges of S_1.

(b) Fractional edges of S_2.

(c) Fractional edges of S_3.

(d) Fractional edges of S_4.

Figure 3.9: Fractional edges of S_1, S_2, S_3, and S_4. In general S_i has a similar arrangement to S_1 and S_3 if i is odd. If i is even, the arrangement is similar to S_2 and S_4.

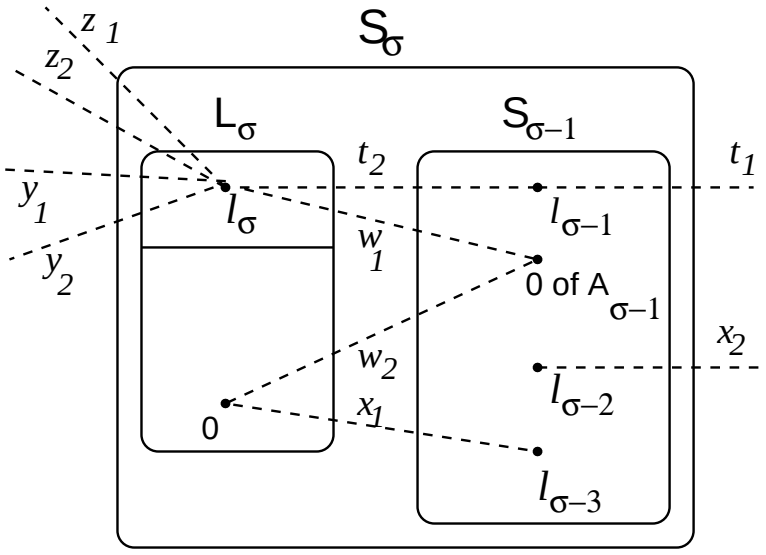


Figure 3.10: Fractional edges of S_σ. If $\sigma = 2$, $\ell_{\sigma-2} = t$ and $\ell_{\sigma-3} = s$.

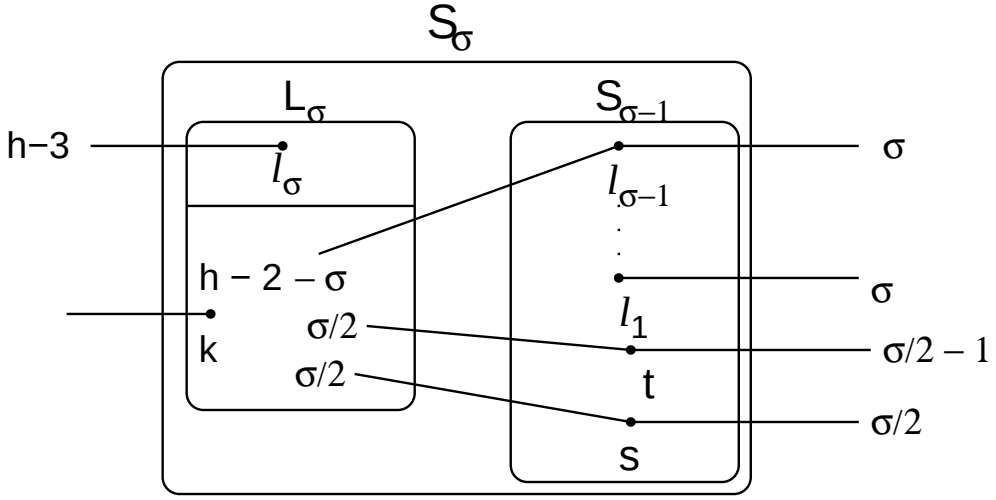


Figure 3.11: Unit edges of S_σ.

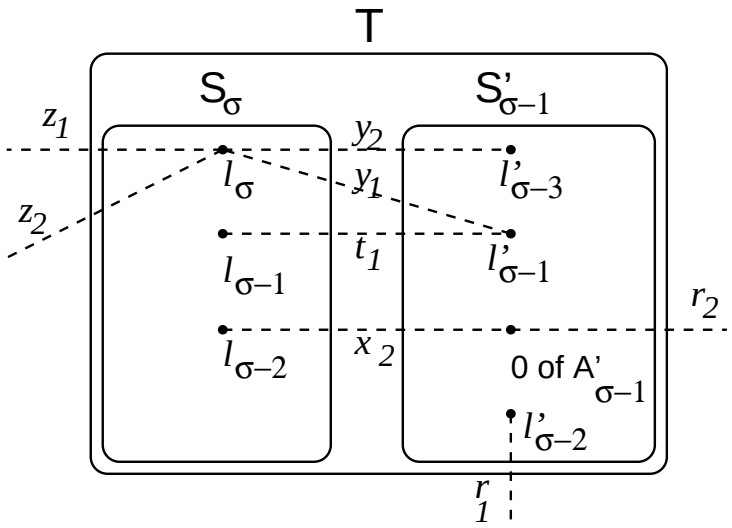


Figure 3.12: Joining S_σ and $S'_{\sigma-1}$ to get a T-set. If $\sigma = 2$, $\ell_{\sigma-2} = t$, $\ell'_{\sigma-2} = t'$, and $\ell'_{\sigma-3} = s'$.

From the equations for S_σ and $S'_{\sigma-1}$, we know

$$r_1 + r_2 = x_2 + t_1 = t_1 + y_1 = x_2 + r_2 = 1$$

and

$$y_1 + y_2 + z_1 + z_2 = 2$$

Making $d(T) = k$ will force $r_1 + r_2 + z_1 + z_2 = 2$ which implies $z_1 + z_2 = 1$ and $y_1 + y_2 = 1$. Combining with previous equations give us

$$r_1 = \overline{r}_2 = \overline{t}_1 = x_2 = y_1 = \overline{y}_2$$

We will take a prime number $p > h+1$ of T-sets and form cycles between them using the unit and fractional edges to complete the example. An example of the edges from a particular T-set is shown in Fig. 3.14. The cycles will have "jumps" of $1, 2, \ldots, h/2, h/2+1$. Number the T-sets from 0 to $p-1$ and arrange the edges from a set T_i as follows: the $h-2$ unit edges from $\ell'_{\sigma-1}$ will go to the $\ell'_{\sigma-1}$ of T_{i+j} and T_{i-j} for $1 \leq j \leq \frac{h-2}{2}$. For the unit edges from ℓ_σ, $h-4$ of them will go to the ℓ_σ of T_{i+j} and T_{i-j} for $1 \leq j \leq \frac{h-4}{2}$. The remaining edge will go the A_σ of $T_{i+\frac{h-2}{2}}$. The unit edge from A_σ will go to the ℓ_σ of $T_{i-\frac{h-2}{2}}$. The unit edge from s will go to the s' of $T_{i+\frac{h}{2}}$, and the unit

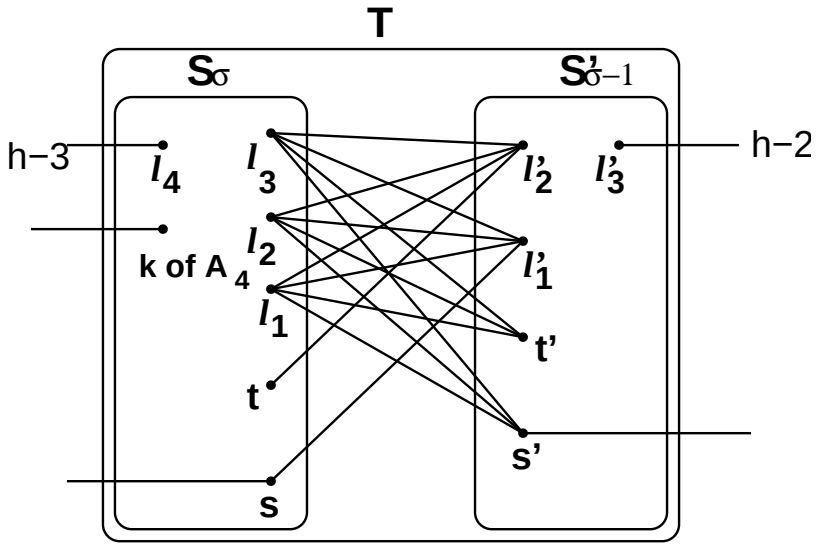


Figure 3.13: The unit edges of a T-set if $\sigma = 4$.

edge from s' will go to the s of $T_{i-\frac{h}{2}}$. The fractional edges from ℓ_σ will go to the ℓ_σ of $T_{i+\frac{h+2}{2}}$ and $T_{i-\frac{h+2}{2}}$. The fractional edge from $\ell'_{\sigma-2}$ will go to the 0 vertex of $A'_{\sigma-1}$ in $T_{i+\frac{h+2}{2}}$ and the fractional edge from the 0 vertex of $A'_{\sigma-1}$ will go to the $\ell'_{\sigma-2}$ of $T_{i-\frac{h+2}{2}}$. All arithmetic on the subscripts of the T-sets is done mod p.

3.3.3 Proof of Uniqueness and k-Connectivity

In order to prove that this is a basic feasible solution, we must prove two things: first, that the values on the fractional edges are unique, and second, that the graph is k-edge-connected.

Proving that the fractional edges are unique is easy due to the invariants. Because the fractional edges between the T-sets are arranged in odd cycles of complementary edges, they are all forced to have weight $\frac{1}{2}$. Once it is determined that the fractional edges between the T-sets have weight $\frac{1}{2}$, the invariants and our equations for the T-set will force all fractional edges in the construction to have weight $\frac{1}{2}$.

We prove that the solution is k-edge-connected the same way that we did in the previous section. First, we show that for any $v \in B_0$, there are k edge-disjoint paths from v to t so we can contract S_0 into a single vertex. Next we show that from any ℓ_i there are k edge-disjoint paths to ℓ_{i+1} so we can contract all ℓ_i into a single vertex ℓ. Then we will show that from S_0 there are k paths to ℓ and that there are k paths from A_i to the contracted $S_0 \cup \ell$. Finally, we show that there are k paths from S_σ to $S'_{\sigma-1}$ and that there are k paths between the T-sets. We will be referencing the paths shown in Fig. 3.3-3.5 of Section 3.2. While most of the paths present in the earlier example will also be present here, not all of them will be. In particular, we will have to be careful about paths that reference "the other $S_{\sigma-1}$" because now the two S-sets that compose our T-set are not symmetric.

3.3.3.1 Paths from B_0 to t

For any $v \in B_0$, $k-2$ of the paths shown in Fig. 3.3 of Section 3.2 are also present in this example. The 2 missing paths come from (b) and (d). The missing path from (b) is because there

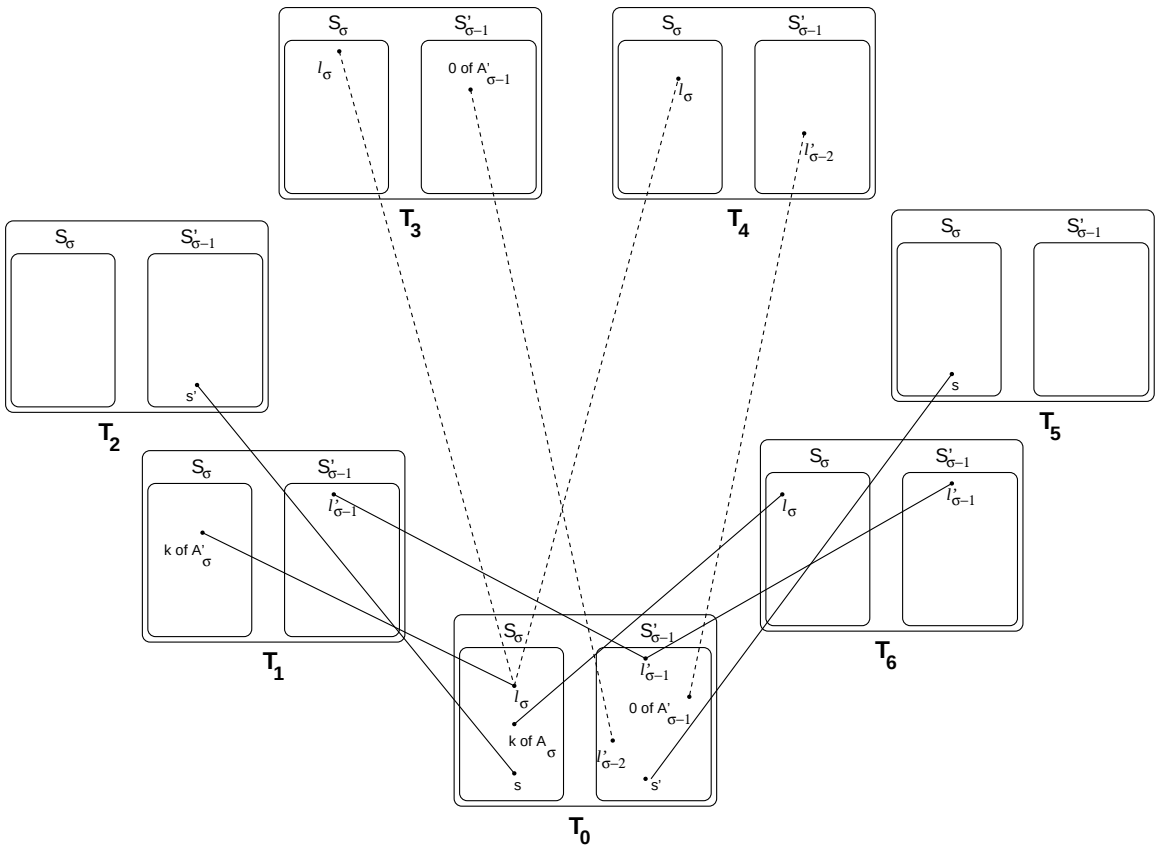


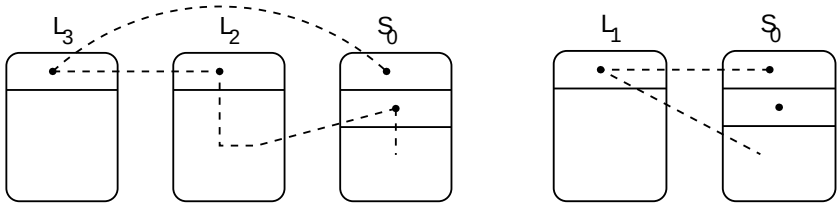
Figure 3.14: The edges from T_0 if $h = 4$ and $p = 7$.

are now only $h-1$ edges between S_0 and A_1. In $S'_{\sigma-1}$ there is 1 fewer path from (d) because there are only $\sigma-1$ vertices in the other set that have an edge to s and t. In S_σ there are $\frac{\sigma+2}{2}$ fewer paths from (d) because there are only $\sigma-2$ vertices in the other set that have an edge to either s or to t. However, we have an additional $\frac{\sigma}{2}$ paths like those in (c), so we are still only 2 paths short. We can replace these paths with the 3 additional paths shown in Fig. 3.15. Two of these paths use fractional edges and hence each only count as $\frac{1}{2}$ of a path. For the path in Fig. 3.15(c), if we are in T_j, the path goes to the s' of $T_{j+\frac{h}{2}}$ then to ℓ_1, A_1 and s. From there it goes to the s' of T_{j+h}, through T_{j+h} to s, to the s' of $T_{j+\frac{3h}{2}}$, etc. until it reaches the s of $T_{j-\frac{h}{2}}$, from which it returns to the s of the initial T-set. Note that because this path goes through all p T-sets, we need to keep track of the path within the T-sets to be sure that no edges are duplicated in other paths (see below).

If $\sigma=2$, the $\frac{1}{2}$ path shown in Fig. 3.15(a) may not exist. See Fig. 3.16 for paths in this case. In Fig. 3.16(a), the path through the other T-set goes to that T-set's ℓ_2 to ℓ'_1 to t' and then to the A'_1 of our original T-set. In Fig. 3.16(b), the path through the other T-set goes to that T-set's ℓ_2, to ℓ'_1 to A'_1 and then to the t' of our original T-set. Note that none of these edges are the same as the ones used in the paths through the other T-set in Fig. 3.15(c).

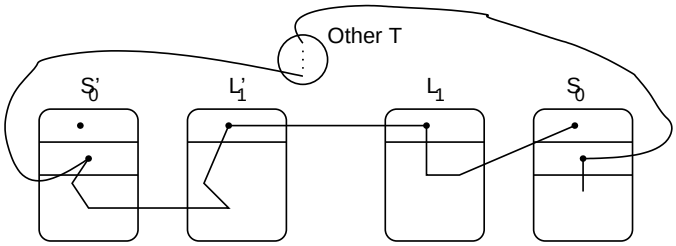
3.3.3.2 Paths from ℓ_i to ℓ_{i+1}

For any ℓ_i, $1 \le i \le \sigma-1$, or ℓ'_i, $1 \le i \le \sigma-2$, all paths shown in Fig. 3.4 of Section 3.2 are also present in this example except for 2 paths from (a) (because there are only $h-\sigma-2$ edges between ℓ_i and A_{i+1}) and some number of paths from (e). If we are in $S'_{\sigma-1}$ 2 paths from (e) are not present because there are only $2\sigma-2$ edges between S'_0 and S_σ. This gives a total of 4 paths that need to be replaced. If we are in S_σ, $\sigma+2$ paths from (e) are not present because there are only $\sigma-2$ edges between S_0 and $S'_{\sigma-1}$. However, there are an additional σ paths like those shown in (b), so we are still only 4 paths short. Note that all paths in (c) and (f) are present. Despite the fact that the 2 maximal S-sets are not symmetric, any ℓ_i in either set will have σ edges to a vertex in the other maximal S-set in the same T-set. Note, however, that if $\ell_i = \ell_{\sigma-1}$ or $\ell'_{\sigma-2}$, the paths



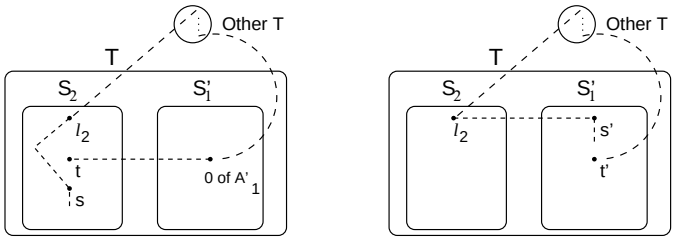
(a) $\frac{1}{2}$ path. If $\sigma = 2$, see Fig. 3.16.

(b) $\frac{1}{2}$ path.



(c) 1 path.

Figure 3.15: 2 additional paths between B_0 and t.



(a) $\frac{1}{2}$ path between B_0 and t in the case $\sigma = 2$.

(b) $\frac{1}{2}$ path between B_0' and t' in the case $\sigma = 2$.

Figure 3.16: Additional paths between B_0 and t when $\sigma = 2$.

in Fig. 3.4(c) go through other T-sets. The replacements for the 4 missing paths differ depending on whether i is even or odd and are shown in Fig. 3.17 (for even values of i) or Fig. 3.18 (for odd values). Note that in the paths in Fig. 3.17(b) and Fig. 3.18(b) L_j is the last L-set the S-set, either L_σ or $L_{\sigma-1}$.

As in Section 3.2.3 we have several facts we can use to validate the paths shown in the figures. The single unit edge between A_{2j} and A_{2j+1} will be used exactly once for each ℓ_i, either in Fig. 3.17(a) or Fig. 3.18(b). There are h paths that use the $h-1$ unit edges and 2 fractional edges between A_{i+1} and ℓ_{i+1}, $h-2$ in Fig. 3.4(a) and (d) and 2 in Fig. 3.17 and Fig. 3.18. In the case $i=1$, there are $h-1$ paths that use the $h-1$ unit edges between A_{i+1} and ℓ_{i+1}, $h-2$ in Fig. 3.4(b), (d), and (e), and 1 from Fig. 3.18. For any $j>1$, the $h-\sigma-2$ edges from L_j to ℓ_{j-1} occur $\leq \sigma(\sigma-1)-2 = h-\sigma-2$ total times in the paths of Fig. 3.4(b), (e), and (f). Similarly for Fig. 3.4(e) and (f), there are $h-\sigma-2$ edge-disjoint paths from $\{s', t', \ell'_g : 1 \leq g \leq \sigma-2\}$ to $\ell'_{\sigma-1}$ in $S'_{\sigma-1}$ and $h-\sigma-2$ edge-disjoint paths from $\{s, t, \ell_g : 1 \leq g \leq \sigma-1\}$ to ℓ_σ in S_σ. This can be increased to $h-2$ edge-disjoint paths if we include the σ edge-disjoint paths from $\{s,t\}$ through A_σ to ℓ_σ or from $\{s',t'\}$ through $A'_{\sigma-1}$ to $\ell'_{\sigma-1}$. This increase allows us to have $h-2$ paths going through the other T-sets in the case $i=\sigma-2$. If $\ell_i = \ell_{\sigma-1}$, the paths in Fig. 3.4(c) go into $S'_{\sigma-1}$, then to $\ell'_{\sigma-1}$ and to another T-set. If $\ell_i = \ell'_{\sigma-2}$, $\sigma-1$ of the paths in Fig. 3.4(c) go into S_σ, then to ℓ_σ and to another T-set. The remaining path goes into S_σ, then to A_σ and to another T-set.

As in Section 3.2.3, for the paths that go through other T-sets, we will give the start and end of the path. These paths will be disjoint because with one exception, a path from a T-set T_j will go through just one other T-set, and that T-set will be part of the collection $\{T_{j+i} : 1 \leq |i| \leq \frac{h-2}{2}\}$. The exceptional path goes through T-sets not part of that collection. Because the T-sets are connected, the specifics of the paths through the other T-sets are not relevant.

For a path through the "other T-set", as referenced in Fig. 3.4(e) and (f), if our original ℓ_i was in S_σ, the path goes to that T-set's $\ell'_{\sigma-1}$, through the T-set to ℓ_σ, and to the ℓ_σ of our initial T-set. If our original ℓ_i was in S'_σ, the path through the other T-set is reversed. If $\ell_i = \ell_{\sigma-1}$, $\sigma-1$ of the paths in Fig. 3.4(c) go to the $\ell'_{\sigma-1}$ of another T-set, through the T-set to ℓ_σ, and to the ℓ_σ

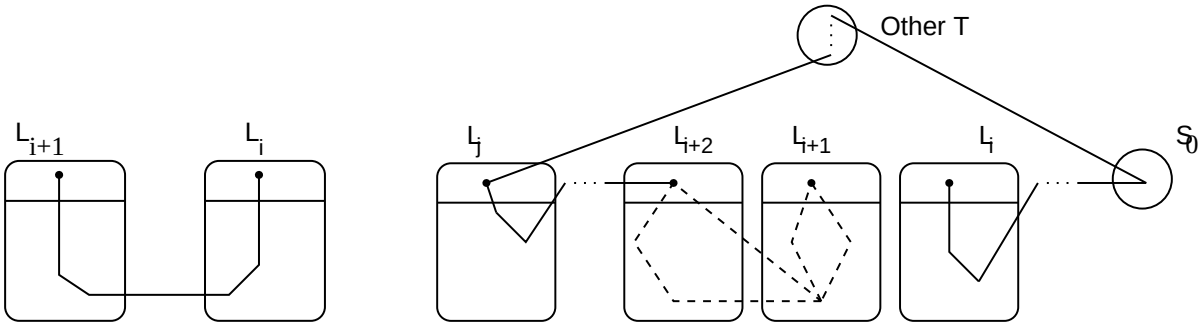
of our initial T-set. The remaining path goes to the $\ell'_{\sigma-1}$ of another T-set, through the T-set to A_σ, and to the ℓ_σ of our initial T-set. If $\ell_i = \ell'_{\sigma-2}$, the paths go to the ℓ_σ of another T-set to $\ell'_{\sigma-1}$, and then to the $\ell'_{\sigma-1}$ of our original T-set.

The exceptional path is the path from either Fig. 3.17(b) or Fig. 3.18(b). If we start in S_σ of T_j and $i \neq \sigma - 1$, the path through goes to $T_{j+\frac{h}{2}}$'s S'_0, to $T_{j+\frac{h}{2}}$'s ℓ_σ, to $T_{j+\frac{h-2}{2}}$'s ℓ_σ, through $T_{j+\frac{h-2}{2}}$ to A_σ, and from there to the ℓ_σ of our initial T-set. If $i = \sigma - 1$, the path is shown in Fig. 3.19. This path goes to $T_{j+\frac{h}{2}}$'s S'_0 to $T_{j+\frac{h}{2}}$'s ℓ_σ to $T_{j+\frac{h+2}{2}}$'s ℓ_σ. From there, the path splits into two $\frac{1}{2}$ paths. The first goes from $T_{j+\frac{h+2}{2}}$'s ℓ_σ back to the ℓ_σ of the original T-set. The second path goes to $T_{j+\frac{h+4}{2}}$'s ℓ_σ, to $T_{j+\frac{h+6}{2}}$'s ℓ_σ, etc. until it reaches $T_{j-\frac{h+2}{2}}$'s ℓ_σ, from which it travels back to the ℓ_σ of our original T set. Note that if $h = 4$, in each T-set the path must go from the ℓ_σ of the current T to the next T-set's A_σ and through that T-set to ℓ_σ. Note that as per the invariant, this path travels through T-sets $T_{j+\frac{h}{2}}$, $T_{j+\frac{h+2}{2}}$, $T_{j+\frac{h+4}{2}}$, $T_{j+\frac{h+6}{2}}$, $\ldots$, $T_{j-\frac{h+2}{2}}$, which are not part of the collection of sets used by the other paths.

If we start in $S'_{\sigma-1}$ of T_j and $i \neq \sigma - 2$, the path goes to S_0 of $T_{j+\frac{h}{2}}$, to $T_{j+\frac{h}{2}}$'s $\ell'_{\sigma-1}$, to $T_{j+\frac{h-2}{2}}$'s $\ell'_{\sigma-1}$ to the original $\ell'_{\sigma-1}$. If $i = \sigma - 2$, the path goes to S_0 of $T_{j+\frac{h}{2}}$, to $T_{j+\frac{h}{2}}$'s ℓ_σ, to $T_{j+\frac{h+2}{2}}$'s ℓ_σ. From there, the path splits into two $\frac{1}{2}$ paths. The first goes from $T_{j+\frac{h+2}{2}}$'s ℓ_σ back to the ℓ_σ of the original T-set, then goes to $\ell_{\sigma-1}$ and to $\ell'_{\sigma-1}$. The second path goes to $T_{j+\frac{h+4}{2}}$'s ℓ_σ, to $T_{j+\frac{h+6}{2}}$'s ℓ_σ, etc. until it reaches $T_{j-\frac{h+2}{2}}$'s ℓ_σ. From there, it goes to $T_{j-\frac{h+2}{2}}$'s $\ell'_{\sigma-2}$, from which it travels back to the $A'_{\sigma-1}$ of our original T set, and from there to $\ell'_{\sigma-1}$.

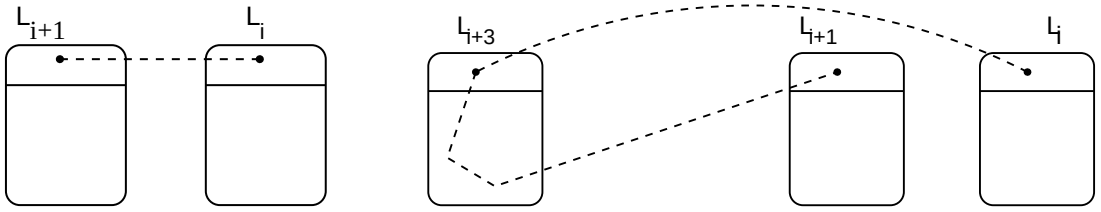
There are some special cases for these figures. When $i \leq 3$ or $i \geq \sigma - 3$, some of the paths shown in Fig. 3.17 and 3.18 may not exist. See Fig. 3.20-3.22 for paths in these cases. In the path in Fig. 3.20(b), the path through the other T-set goes to that T-set's $A'_{\sigma-1}$ to ℓ_σ to the ℓ_σ of our original T-set. The path in Fig. 3.22(d) is reversed from the path for Fig. 3.20(b).

Note that for any ℓ_i, there are at most h distinct paths through other T-sets, $h - 2$ from Fig. 3.4 and 2 from Fig. 3.17-3.22. Therefore, we have enough T-sets that each path can go through a different T-set.



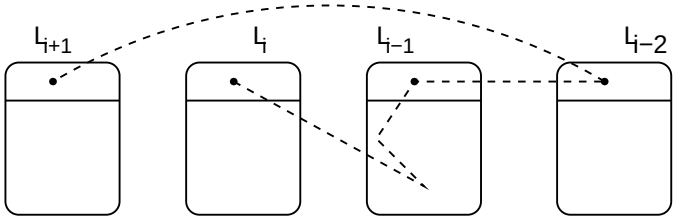
(a) 1 path.

(b) 1 path (in the dashed parts, path splits into two $\frac{1}{2}$ paths). L_j is the last L-set in this S-set.

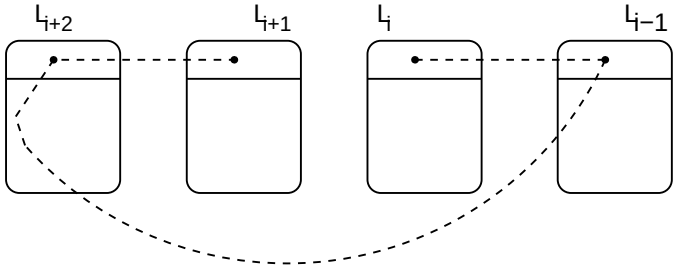


(c) $\frac{1}{2}$ path.

(d) $\frac{1}{2}$ path. If $i = \sigma - 2$, see Fig. 3.20(b) and (c).

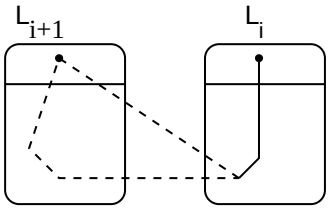


(e) $\frac{1}{2}$ path. If $i = 2$, see Fig. 3.20(d).

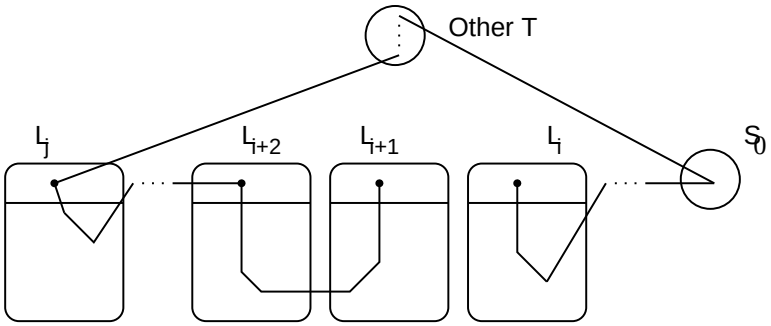


(f) $\frac{1}{2}$ path. If $\ell_i = \ell'_{\sigma-2}$, see Fig. 3.20(a).

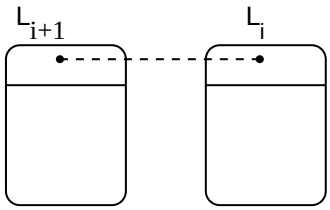
Figure 3.17: 4 additional paths between ℓ_i and ℓ_{i+1} for even values of i.



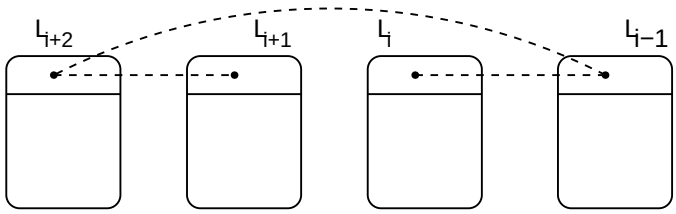
(a) 1 path (in the dashed part, path splits into two $\frac{1}{2}$ paths).



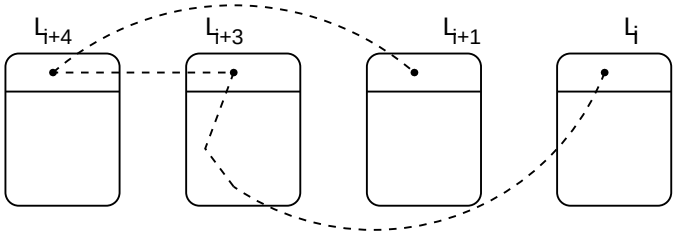
(b) 1 path. L_j is the last L-set in this S-set.



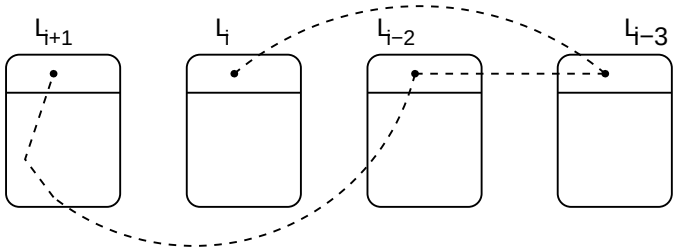
(c) $\frac{1}{2}$ path.



(d) $\frac{1}{2}$ path. If $i = 1$ or $\sigma - 1$, see Fig. 3.21(b) and (b).



(e) $\frac{1}{2}$ path. If $i \geq \sigma - 3$, see Fig. 3.22.



(f) $\frac{1}{2}$ path. If $i \leq 3$, see Fig. 3.21(a) and (c).

Figure 3.18: 4 additional paths between ℓ_i and ℓ_{i+1} for odd values of i.

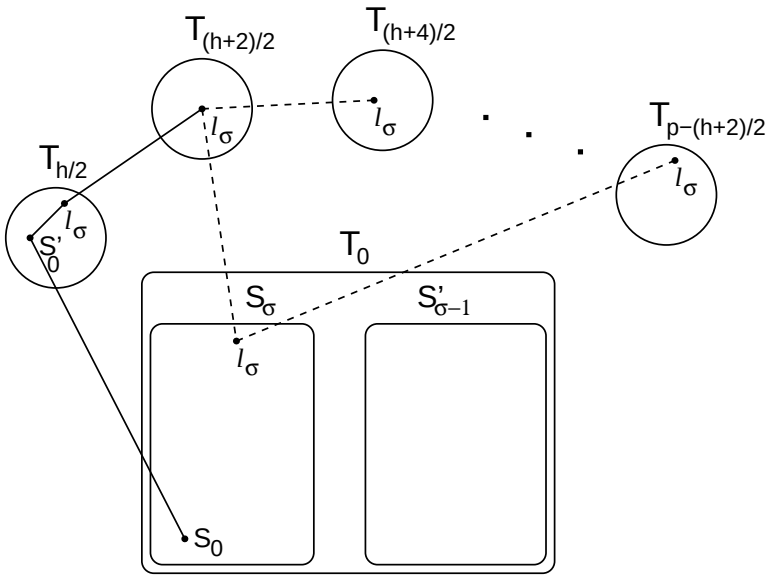
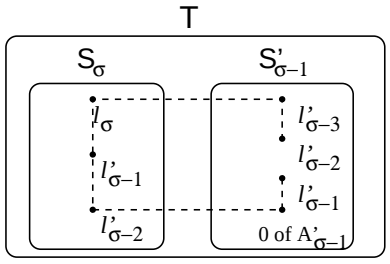
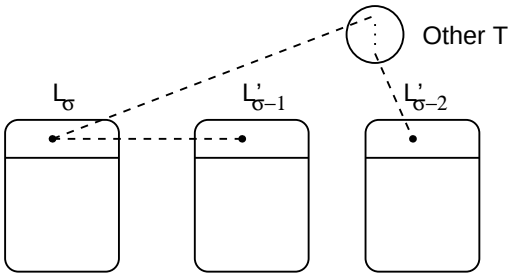


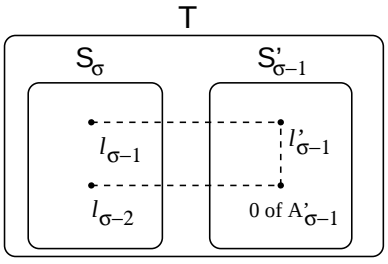
Figure 3.19: The path through another T-set used in Fig. 3.18(b) when $i = \sigma - 1$. A similar path is used through the other T-set in Fig. 3.17(b) when $i = \sigma - 2$. If $h = 4$, in each T-set the path must go from the ℓ_σ of the current T to the next T-set's A_σ and through that T-set to ℓ_σ.



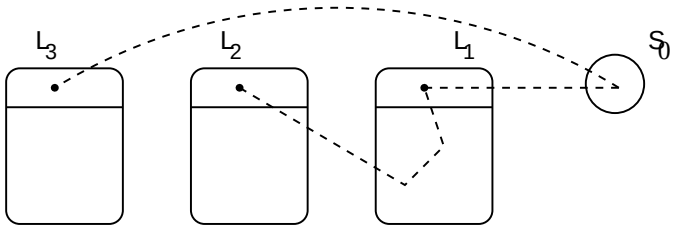
(a) Additional $\frac{1}{2}$ path in the case $\ell_i = \ell'_{\sigma-2}$.



(b) Additional $\frac{1}{2}$ path in the case $\ell_i = \ell'_{\sigma-2}$.

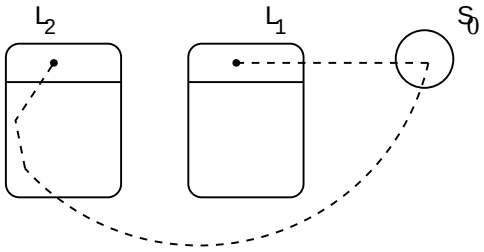


(c) Additional $\frac{1}{2}$ path in the case $\ell_i = \ell_{\sigma-2}$.

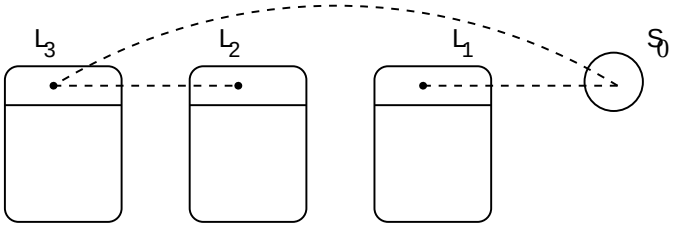


(d) Additional $\frac{1}{2}$ path if $i = 2$.

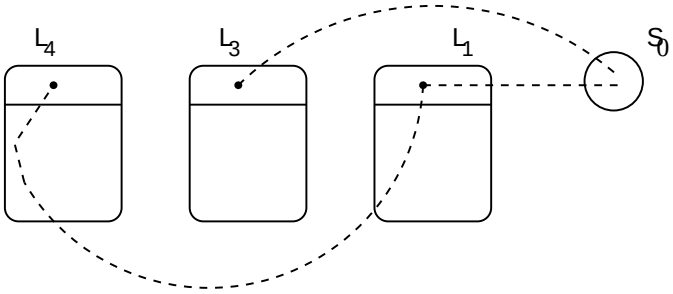
Figure 3.20: Additional paths between ℓ_i and ℓ_{i+1} for even values of i.



(a) Additional $\frac{1}{2}$ path in the case $i = 1$.

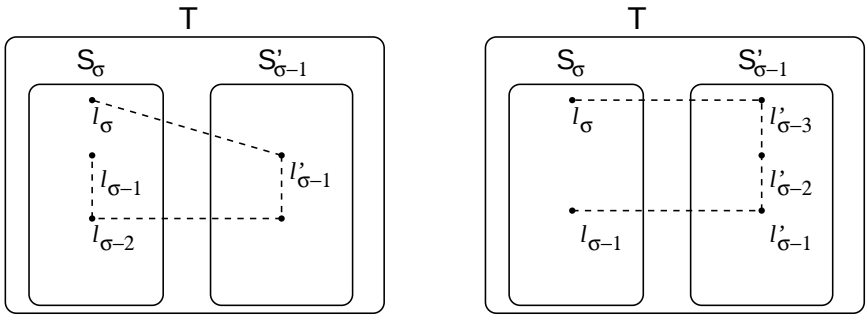


(b) Additional $\frac{1}{2}$ path in the case $i = 1$.



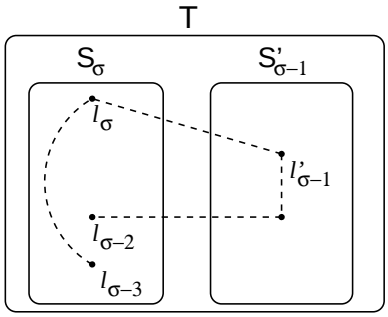
(c) Additional $\frac{1}{2}$ path in the case $i = 3$.

Figure 3.21: Additional paths between ℓ_i and ℓ_{i+1} for odd values of $i \leq 3$.

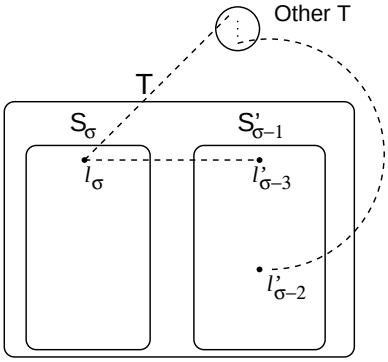


(a) Additional $\frac{1}{2}$ path in the case $\ell_i = \ell_{\sigma-1}$.

(b) Additional $\frac{1}{2}$ path in the case $\ell_i = \ell_{\sigma-1}$.



(c) Additional $\frac{1}{2}$ path in the case $\ell_i = \ell_{\sigma-3}$.



(d) Additional $\frac{1}{2}$ path in the case $\ell_i = \ell'_{\sigma-3}$.

Figure 3.22: Additional paths between ℓ_i and ℓ_{i+1} for odd values of $i \geq \sigma - 3$.

3.3.3.3 Other Paths

For paths between S_0 and ℓ, if we are in S_σ, there are actually $k - \sigma - \frac{1}{2}$ paths like those shown in Fig. 3.5(a) and $\sigma - 2$ like those shown in Fig. 3.5(b). If we are in $S_{\sigma-1}$, there are $k - 2\sigma - \frac{1}{2}$ paths like those shown in Fig. 3.5(a) and $2\sigma - 2$ like those shown in Fig. 3.5(b). In both cases, we need 2 $\frac{1}{2}$ additional paths. There are 1 $\frac{1}{2}$ paths that go from S_0 directly to ℓ. We can find one additional path that goes from S_0 to the S_0' of another T-set, to the ℓ of that T-set, and from there to the ℓ of our original T-set.

For paths between A_i and $\ell \cup S_0$, $k - 2$ of the edges leaving A_i go to either ℓ or S_0. The remaining two edges go to some other A_j from which we can find a path to ℓ.

Just as in the previous example, there are h edges between S_σ and $S'_{\sigma-1}$. There are also h paths between S_σ and $S'_{\sigma-1}$ which use the h cycles in the T-sets. This gives a total of k paths. There are h cycles between the T-sets, insuring that they are k-connected.

3.3.4 Approximation Ratio

To calculate the approximation ratio for this example, we first note that there is a 1-1 correspondence between sets in the laminar family and edges with weight $\frac{1}{2}$. We can show this correspondence by noting that each set contributes 2 to the total fractional degree. By the handshaking lemma, this implies that the number of sets and fractional edges is the same. Each vertex is a singleton set and has fractional degree at least 2. Each of A_i, $0 \leq i \leq \sigma$, B_0, and S_0 has one vertex of fractional degree 4 that is not in any smaller non-singleton set, so each of those sets contributes an additional 2 to the fractional degree beyond what was already counted for the vertices. Each L_i, $1 \leq i \leq \sigma - 1$, has a vertex of fractional degree 6, which will contribute an additional 4 to the fractional degree: 2 for L_i and 2 for S_i. Finally, L_σ has a vertex of fractional degree 8, which will contribute an additional 6 to the fractional degree: 2 for L_σ, 2 for S_σ, and 2 for T. Therefore, we can count the number of fractional edges by counting the number of sets in the laminar family.

As before, we calculate the size of $\mathcal{L}$ in terms of the number of vertices in the graph, n, by

calculating the number of vertices, n_T, and non-singleton sets, a, in each T-set. Each T-set contains an S_σ and an $S_{\sigma-1}$. This means each T-set contains $2\sigma-1$ L-sets, which each contain $k+2$ vertices. Each T-set also contains 2 S_0, which each contain $k+3$ vertices. Thus $n_T = (2\sigma+1)(k+2)+2$. Now, each of the $2\sigma-1$ L-sets contributes 2 nonsingleton sets. Each of the 2 S_0 contributes 3 nonsingleton sets. There are $2\sigma-1$ larger S-sets, as well as the T-set itself. Therefore

$$a = 2(2\sigma-1)+6+(2\sigma-1)+1 = 6\sigma+4.$$

This gives us a total size for $\mathcal{L}$ of

$$n(1+\frac{(6\sigma+3)+1}{(2\sigma+1)(k+2)+2}) = n\left(1+\frac{3}{k}+\frac{1}{2\sigma k}-O(\frac{1}{k^2})\right).$$

When we substitute for $\sigma = \sqrt{h} = \sqrt{\frac{k}{2}}$, we get $n\left(1+\frac{3}{k}+\frac{1}{\sqrt{2}k\sqrt{k}}-O(\frac{1}{k^2})\right)$ as the number of critical sets and thus the number of fractional edges. By rounding up all the edges of weight $\frac{1}{2}$, we get an approximation ratio of $1+\frac{1}{k}+\frac{3}{k^2}+\frac{1}{\sqrt{2}k^2\sqrt{k}}-O(\frac{1}{k^3})$.

Chapter 4

A Bound on Special Edges

In this chapter we turn our attention to directed graphs and a combinatorial as opposed to linear programming-based algorithm for approximating the minimum k-ECSS. We extend Gabow's bound on the number of special edges that can be in a k-connected graph. In Section 4.1, we go over the details of Gabow's proof necessary to understand our work. We also prove some facts about the size of tricritical sets. In Section 4.2 we give the proof for our extension to $10 \leq k < 15$.

Recall from Chapter 2 that a critical edge e of a k-connected graph G is one such that $G-e$ is not k-connected, and a special edge is a critical edge whose endpoints both have degree greater than k. How many special edges can there be in a k-connected graph with n vertices? A bound on the number of special edges is an interesting combinatorial problem in its own right, but such a bound also plays a role in the analysis of the Cheriyan and Thurimella [1] algorithm described in Chapter 2. The Cheriyan and Thurimella algorithm has an approximation bound that explicitly depends on the number of special edges. The algorithm is guaranteed to approximate the minimum k-ECSS in a simple directed graph to within $1 + \frac{s}{kn}$, where s is the number of special edges. Cheriyan and Thurimella showed that a simple directed graph will have no more than $4\sqrt{k}n$ special edges. While the Cheriyan and Thurimella algorithm has an approximation bound that has been eclipsed by subsequent linear programming algorithms, it is still the best known combinatorial algorithm. Linear programming algorithms are expensive in terms of both time and space. The Cheriyan and Thurimella algorithm is much faster and requires far less space.

Gabow [13] improved the upper bound on the number of special edges that can be in a simple

directed graph from Cheriyan and Thurimella's bound of $4\sqrt{kn}$ to a number slightly larger than $\sqrt{2kn}$ for values of k greater than or equal to 15. He also gave a lower bound example for all values of k greater than or equal to 1, proving that the upper bound was tight for all values of $k \geq 15$. In addition he proved an upper bound of $5n$ for values of k greater than or equal to 6 with the caveat that this bound is certainly loose.

Here, we extend the $\sqrt{2kn}$ bound to values of k between 10 and 15, improving on the previous $5n$ bound. Because of the lower bound example given in [13], we know that this bound is tight. All references to "graphs" in this chapter refer to directed graphs.

4.1 An Upper Bound on Special Edges for $k \geq 15$

This section gives a summary of the proof of Gabow's bound on the number of special edges for $k \geq 15$. All results in Section 4.1 come from Gabow, "Special Edges, and Approximating the Smallest Directed k-connected Spanning Subgraph" [13] unless otherwise noted.

For every value of k, define two integers τ and ω such that $0 \leq \omega \leq \tau$ and:

$$k = \frac{\tau(\tau+1)}{2} + \omega \tag{4.1}$$

Note that for a given value of k, τ and ω are unique. In order to see this, consider the "triangle numbers," those numbers $T_n = \sum_{i=0}^{n} i$. For each value of k, the values of τ and ω are the unique values such that $T_\tau \leq k < T_{\tau+1}$ and $k = T_\tau + \omega$.

Once the values of τ and ω are determined, we can then define a function σ on k such that:

$$\sigma(k) = \tau + \frac{\omega}{\tau+1} \tag{4.2}$$

From here on, the value of k will be clear from context, so we will refer to this number as σ.

Gabow proved that for $k \geq 15$, there will be no more than $n\sigma$ special edges in a k-edge-connected graph. We give some of the details of the proof that will later be used in our extension to $10 \leq k < 15$.

4.1.1 Criticality

Say that two critical sets T_1 and T_2 are **same-way critical** if both sets are in-critical or both sets are out-critical. Likewise, say that T_1 and T_2 are **opposite-way critical** if T_1 is in-critical and T_2 is out-critical or vice versa. A set is **two-way critical** if it is both in-critical and out-critical.

Lemma 2 gives a general fact about in- and out- critical sets. It appears in [12].

Lemma 2. *Let $\{Z_1, ..., Z_{j-1}\}$ be a collection of sets that are all same-way critical and W be a set that is opposite-way critical from the Z_i. If $W = \bigcup_{i=0}^{j-1} Z_i$, then W and all Z_i are 2-way critical.*

Proof. WLOG, assume that W is out-critical and the Z_i are in-critical. We will show that W is also in-critical and the Z_i are out-critical. For any set S, let $\rho(S)$ be the in-degree of S and $\delta(S)$ be the out-degree of S. Let b be the number of edges that go between the Z_i. Then, the in-degree of W is $\rho(W) = \sum_{i=0}^{j-1} \rho(Z_i) - b$ and the out-degree of W is $\delta(W) = \sum_{i=0}^{j-1} \delta(Z_i) - b$. Combining these two equations gives

$$\rho(W) + \sum_{i=0}^{j-1} \delta(Z_i) = \delta(W) + \sum_{i=0}^{j-1} \rho(Z_i)$$

We were given $\delta(W) = \rho(Z_i) = k$, so this reduces to

$$\rho(W) + \sum_{i=0}^{j-1} \delta(Z_i) = (j+1)k$$

Because the graph is k-edge-connected, all sets must have in- and out-degree at least k, so $\rho(W) \geq k$ and $\delta(Z_i) \geq k$. The only way that this can be consistent with the the previous equation is if $\rho(W) = \delta(Z_i) = k$ □

Recall that in a simple graph, a k-critical set must either have cardinality 1 or a cardinality greater than or equal to k. If we are looking for critical sets that cover special edges, we can expand this somewhat: a critical set that covers special edges must have cardinality greater than or equal to $k+1$. This is because a singleton set that is critical consists of a vertex with degree k, while a critical set with exactly k vertices is a clique on k vertices where every vertex has exactly one edge that leaves the set, so all vertices have degree k and the set cannot cover any special edges.

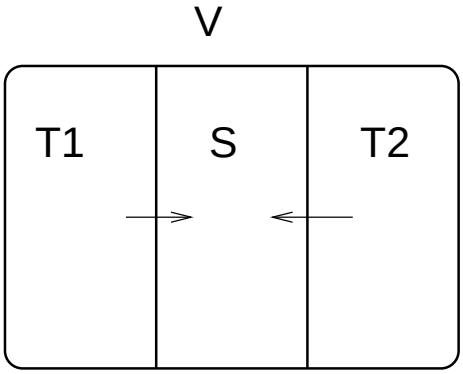


Figure 4.1: An example of an in-bicritical set. If $T1$ and $T2$ are out-critical, S is in-bicritical.

Recall also that a bicritical set is a set S such that V can be partitioned into T_1, T_2, S where T_1 and T_2 are same-way critical. Say that two critical sets T_1 and T_2 in our laminar family **alternate** if $T_1 \subset T_2$ and T_1 and T_2 are opposite-way critical. Note that if T_1 and T_2 alternate, then $T_2 - T_1$ is bicritical (with the partition being T_1, $V - T_2$, $T_2 - T_1$). See Fig. 4.1 for examples of bicritical sets.

From Chapters 2 and 3, recall that for an r-critical set S with s vertices, we know:

$$s(k - s + 1) \leq rk$$

Gabow used this equation to prove several important lemmas about bicritical sets (given here without proof):

Lemma 3. *Let G be a simple k-edge-connected graph and S be a bicritical set with the partition of V being S, T_1, T_2. Then, the following assertions hold:*

(i) If $k \geq 7$, then $|S| \leq 2$ or $|S| \geq k - 1$.

(ii) If $|S| = 2$ or $|S| = k - 1$ then $d(T_1, T_2) \leq 2$.

(iii) If $|S| = k - 1$ then T_1 and T_2 together cover at most 6 special edges.

Lemma 4. *Let G be a simple k-edge-connected graph for $k \geq 7$. Consider critical sets $A \subset B \subset C$. Then, the following assertions hold:*

(i) If B alternates with A and with C then $|C - A| \neq 3, 4$.

(ii) If B alternates with A then $|C - A| \leq 2$ or $|C - A| \geq k - 1$.

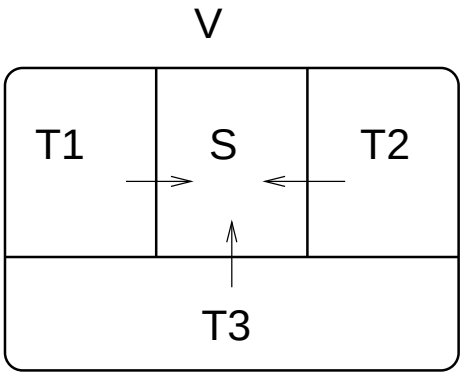


Figure 4.2: An example of an in-tricritical set. If $T1$, $T2$, and $T3$ are out-critical, S is in-tricritical.

The same ideas used to prove these lemmas about bicritical sets can be extended to tricritical sets. Recall that a tricritical set is a set S such that V can be partitioned into T_1, T_2, T_3, S where T_1 and T_2 are same-way critical. See Fig. 4.2 for examples. Lemmas 5 and 6 are original and not from [13]:

Lemma 5. *If $k \geq 10$ and the vertices of S actually cover any special edges, then $|S| \neq 5$ or 6.*

Proof. Using the equation $|S| = s, s(k - s + 1) \leq rk$, substituting $r = 3$ and $k = 10$ gives $s(11 - s) \leq 30$. This gives exact equality for $s = 5$ or 6. Exact equality implies that all vertices in S have degree exactly k, meaning no special edges can have one of those vertices as an endpoint. □

Lemma 6. *If S is tricritical, with at least 3 special vertices, then $|S| = 3$ or $|S| \geq k - 2$ for $k \geq 10$.*

Proof. Recall from Chapter 2 that a special vertex is one with degree greater than k. Say that there are t special vertices. Modifying our previous equation to account for the special vertices, we have $s(k - (s - 1)) + t \leq 3k$. For $t = 3$ and $s = 4$ (or $s = k - 3$), this gives

$$4(k - 3) + 3 = 4k - 9 \leq 3k$$

which implies $k \leq 9$. □

4.1.2 The laminar family and paying for special edges

As mentioned previously, all critical edges (and therefore all special edges) can be covered by a laminar family of critical sets [8]. Gabow's proof makes use of this fact, examining a single laminar family $\mathcal{L}$ of in-critical and out-critical sets that covers all special edges in the graph. The laminar family can be treated as a tree, where for two sets $T_1, T_2 \in \mathcal{L}$, T_1 is a child of T_2 if $T_1 \subset T_2$ and there is no $S \in \mathcal{L}$ such that $T_1 \subset S \subset T_2$. This tree has three types of nodes: **leaves**, **chain nodes** (those nodes with exactly one child), and **branching nodes** (those nodes with multiple children). Leaves and branching nodes will be referred to collectively as **non-chain nodes**. In order to minimize confusion, for the remainder of this chapter we will use "vertex" to refer to a vertex in the original graph and "node" to refer to a set in the tree representing $\mathcal{L}$.

We count the number of special edges in the following way: give each vertex in the graph σ "credits." Each special edge must be "paid for" with one credit. The theorem is proved by paying for all special edges using the credits on the vertices.

Each leaf in $\mathcal{L}$ has at least $(k+1)\sigma$ credits on its vertices. Take these credits from the leaves and use them to form **shares**, each share containing $\frac{(k+1)\sigma}{2}$ credits. These shares are distributed to the nodes of the tree. Each leaf receives one share. A non-leaf node with C children receives C shares if it is the root or $C-1$ shares if it is not the root. If a branching node receives more than one share of credits, every share beyond the first share is called a **bonus share**.

The shares on the leaf nodes and branching nodes are sufficient to pay for any special edges leaving those sets. The difficult part of the proof comes from paying for the special edges of the chain nodes. This is done using a method called **path payment**.

4.1.2.1 Path payment

We partition $\mathcal{L}$ into paths in the tree. Each path starts with a non-chain node A_0 and includes all consecutive ancestors of A_0 up to A_q that are all chain nodes: A_i is the unique child of A_{i+1} for all $0 \leq i < q$, and A_q's parent is a non-chain node. We further divide the path into subpaths

with boundary nodes B_i. Let $B_0 = A_0$. For $i > 0$, let $B_i = A_j$ if A_j is the first node in the path such that $|A_j - B_{i-1}| > \tau$, or A_q if no such A_j exists. The subpath P_i consists of all nodes that are proper ancestors of B_{i-1} and descendants of B_i (including B_i itself). See Fig. 4.3

For subpaths where $|B_i - B_{i-1}| > \tau$ we can pay for all special edges covered by the subpath using the following lemmas. To simplify notation, let $P_i = P$, $B_{i-1} = A$, and $B_i = Z$. Let Y be the last set before Z (note that it is possible $Y = A$), and let B be the first set of P that alternates with A (with the caveat that it is possible B does not exist). Let the function $Credits(P)$ be the number of credits on the vertices in $Z - A$. Let $Special(P)$ be the number of special edges covered by sets in P but not covered by A or any of its descendants.

We present these lemmas along with a sketch of the proofs. For full proofs, see [13].

Lemma 7. *For all values of k, if $|Z - A| > \tau$ and all sets in $\{A\} \cup P$ are same-way critical, then $Credits(P) \geq Special(P)$.*

Proof. **(Sketch)** Let $|Y - A| = s$ and $|Z - Y| = t$. Then, we could have $\frac{s(s-1)}{2}$ special edges with both ends in $|Y - A|$, st with one end in $|Y - A|$ and the other in $|Z - Y|$, and k incident to Z. See Fig. 4.4. Using the inequality $s \leq \tau < s + t$ we can show $k \leq \frac{s(s+1)}{2} + (\tau - s - \frac{\omega}{\tau+1})(s+t)$. Now we can say

$$Special(P) \leq \frac{s(s-1)}{2} + st + k \leq s(s+t) + (\tau - s - \frac{\omega}{\tau+1})(s+t) = \sigma(s+t)$$

Because $\sigma(s+t) = Credits(P)$, the proof is complete. □

Lemma 8. *For $k \geq 10$, if $|Z - A| > \tau$, B exists, and $|B - A| > 2$, then $Credits(P) \geq Special(P)$.*

Proof. **(Sketch)** The condition on k implies $\tau \geq 4$ and $k > \tau + 4$. Because $|B - A|$ is bicritical, $|B - A| > 2$ implies $|B - A| \geq k - 1$. Because $k - 1 > \tau$, this also implies $B = Z$. This further implies that Y alternates with Z, which along with $|Z - A| \geq k - 1$ implies $|Z - Y| \geq k - 1$. If $|Y - A| = s$, there are at most $\frac{s(s-1)}{2}$ special edges with both ends in $|Y - A|$, k special edges incident to Y, and $2k$ incident to Z. See Fig. 4.5. We also have

$$Credits(P) \geq s\tau + (k-1)\tau > s^2 + 4k - 4 > s^2 + 3k > \frac{s(s-1)}{2} + 3k \geq Special(P)$$

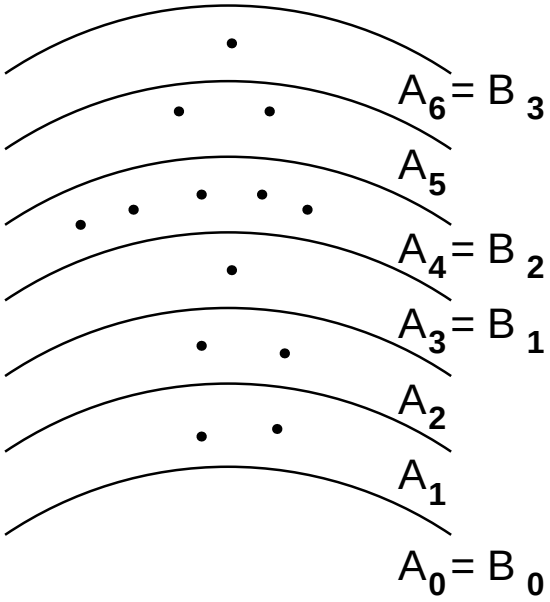


Figure 4.3: The sequence $A_0, \ldots, A_6$ is an example of a path. The parent of A_6 is a non-chain node. If $\tau = 4$, then A_0, A_3, A_4, and A_6 are the boundary nodes dividing the path into subpaths.

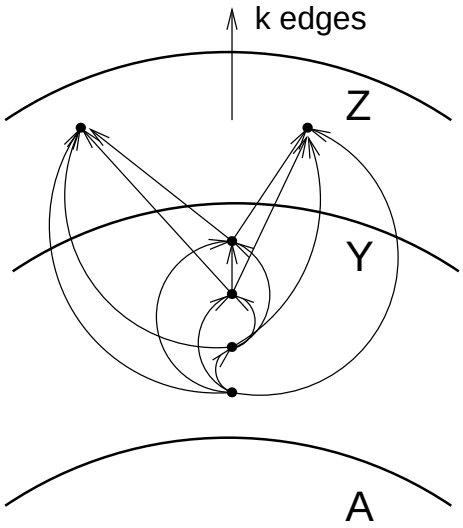


Figure 4.4: Proof of Lemma 7. A subpath with no alternation and the special edges that the sets in the subpath may cover.

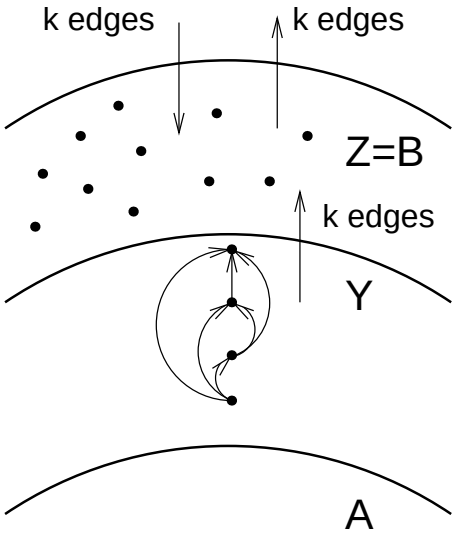


Figure 4.5: Proof of Lemma 8. Possible special edges in a subpath where B is the first set that alternates with A and $|B - A| > 2$ (which implies that $B = Z$, the last set of the subpath).

This completes the proof. □

Lemma 9. *For $k \geq 21$, if $|Z - A| > \tau$, B exists, and $|B - A| \leq 2$, then $Credits(P) \geq Special(P)$.*

Proof. **(Sketch)** The condition on k implies $\tau \geq 6$. Let C be the first set of P with $|C - A| > 2$. C alternates with either A or B, so we can show $|C - A| \geq k - 1$ and therefore $C = Z$. Thus, $|Y - A| \leq 2$, so we have at most two special edges that have both endpoints in Y, k incident to A, and $2k$ incident to each of Y and Z. See Fig. 4.6. We have

$$Credits(P) \geq 6k - 6 > 5k + 2 \geq Special(P)$$

which completes the proof. □

4.1.2.2 Finishing details

All that remains is to pay for the edges covered by the last subpath. Because of the way that the subpaths were defined, the last subpath is the only one where it is possible that $|Z - A| \leq \tau$. We have another lemma that aids in this case.

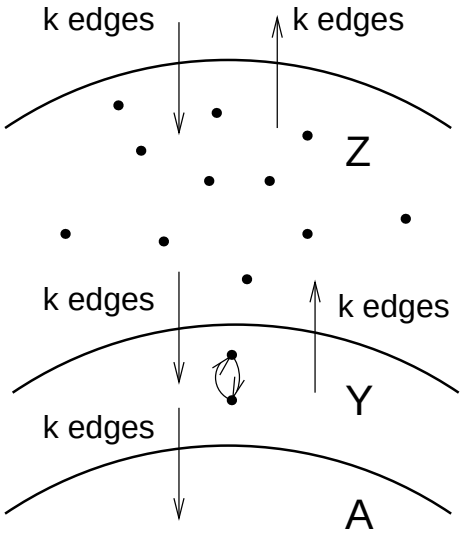


Figure 4.6: Proof of Lemma 9. Possible special edges in a subpath where B is the first set that alternates with A and $|B - A| \leq 2$.

Lemma 10. *For all k, if $|Z - A| \leq \tau$, $Credits(P) \geq Special(P) + 3k$.*

Proof. The fact that $|Z - A| \leq \tau$ implies that $Credits(P)$ is greater than the number of special edges with both ends in $|Z - A|$. There may be k special edges incident to A and $2k$ special edges incident to Z. The lemma follows. □

The difficulty of paying for the remaining $3k$ special edges depends on the value of k. If $k \geq 55$, then A_0 has at least $5k$ credits, enough to pay for its own special edges and the $3k$ remaining special edges of the path. Otherwise, we need to look more closely at our path payment and realize that we overestimated the credits needed to pay for our special edges. Let a set X be saturated if it is either 2-way critical or is 1-way critical and has k unspent credits. Note that:

Lemma 11. *For $k \geq 21$, if $|Z - A| > \tau$ and A is saturated at the beginning of path payment, Z will be saturated at the end of path payment.*

Proof. **(Sketch)** Lemmas 7-9 all pay for special edges in a way that satisfies this condition. Lemma 7 assumes that both A and Z are 1-way critical, so we can just transfer the k credits on A to Z. Lemmas 8 and 9 both allocate $2k$ credits to pay for special edges incident to Z; if Z is 1-way critical, the unused k credits can be used to saturate it. □

This extension to path payment allows us to pay for all special edges if $k \geq 36$. A_0 has at least $4k$ credits, so $2k$ can be used to saturate A_0, and $2k$ can be used to pay for special edges incident to Z in the last subpath. In the last subpath, if A was 2-way critical, there are no additional special edges incident to A. If A was 1-way critical, the extra k credits on A can be used to pay for any special edges incident to it.

If $k < 36$, there may be some edges that are not paid for by path payment. We call these edges **liabilities**, and the following lemma helps pay for them:

Lemma 12. *Let $k \geq 10$. Let P be the last subpath of $A_0, ..., A_q$. If P has liabilities then the following properties hold:*

(i) A, P has an alternation;

(ii) $|Z - A| \leq 2$*;*

(iii) every vertex of $Z - A$ *has* $\geq \sigma + 1 - |Z - A|$ *credits.*

Proof. **(Sketch)** Note that $|Z - A| \leq \tau$, otherwise one of Lemmas 7-9 will let us pay for all special edges. All special edges within P are paid for by Lemma 10. If A, P does not contain an alternation, then all edges incident to A have already been paid for and there are k credits on A because A is saturated. These k credits can be used to pay for the k edges incident to Z. Thus, if P has liabilities, A, P must have an alternation. This alternation also implies $|Z - A| \leq 2$ by Lemma 4**(ii)**. After we pay for all edges in P, **(iii)** holds. □

This lemma and others are used to pay for the cases $k \geq 21$ and $k \geq 15$. We omit the additional lemmas because these lemmas and their proofs do not apply directly to our proof for $10 \leq k < 15$. See [13] for details.

4.2 Extension to $10 \leq k < 15$

In this section we will extend the $n\sigma$ bound on the number of special edges to values of k between 10 and 15. We will prove the following theorem:

Theorem 5. *For* $k \geq 10$*, any* k*-edge-connected graph has no more than* $n\sigma$ *special edges. This bound is tight.*

The fact that the bound is tight comes from the lower bound example in [13]. In order to the upper bound proof for $10 \leq k < 15$, we need to introduce a new method of path payment.

4.2.1 Conservative Path Payment

Conservative path payment is a variation on path payment that allows us to save some of the credits of a non-chain node in order to help pay for the liabilities of its children. For each non-chain node W, let W' be the first ancestor of W that alternates with it ($W = W'$ if W is 2-way critical). If W is 2-way critical or $|W' - W| = 1$, charge W the $2k$ credits necessary to cover

all its entering and leaving liabilities. Otherwise, if $|W' - W| \geq 2$ or W' does not exist, charge W only the k-credits necessary to cover its own liabilities. The other k credits are used to pay for the liabilities of W's children.

The reason that we use conservative path payment to pay for the special edges when $k < 15$ is that, when $W' - W$ is large, it is difficult to pay for the liabilities of W's children that are incident to W. Conservative path payment helps offset this by giving us k additional credits to help pay for these liabilities.

Previously, we charged W $2k$ credits in order to saturate it. This allowed us to saturate A, the set before our final subpath. Under conservative path payment, A will still be saturated except in the case that all sets on the path are same-way critical.

4.2.2 Modified Lemmas

There are several lemmas that need to be modified from the previous section. First we give a lemma that will help us in extending the lemmas from Section 4.1.

Lemma 13. *Let $k \geq 10$. If B is the first set that alternates with A_0 and $|B - A_0| > \tau$, then either B or the last set in the subpath immediately before B will be saturated.*

Proof. If B is the last set of a subpath, Lemma 8 applies. Otherwise, let Z be the last set on the subpath before B and note that this implies $|B - Z| \leq 2$. Let Y be the set before Z. Because B is the first set that alternates with A_0, the subpath before B does not contain an alternation. Thus B alternates with Z, Y, and A.

Since $|Z - A| > \tau$ by definition, $|B - A| > \tau$ and thus $|B - A| \geq k - 1$. This implies that since $|Y - A| \leq \tau$, $|B - Y| \geq k - 1$. Because $|B - Z| \leq 2$, therefore $|Z - Y| \geq k - 3$.

The special edges with both ends in $Y - A$ can be paid for using the credits on the vertices of $|Y - A|$, leaving at least 4 credits. There are at least $4k - 12$ credits on the vertices of $Z - Y$. When these are added to the 4 credits remaining on the vertices of $|Y - A|$, this gives a total of $4k - 8 > 3k$ credits. This is enough to pay for the at most k special edges covered by Y and saturate

Z. □

Now we are ready to modify the lemmas from Section 4.1 in order to use conservative path payment and extend the entire proof to $10 \leq k < 15$. As in Section 4.1.2.1, let A be the first set of a subpath P, Z be the last set of P, Y the last set before Z, and B the first set of P that alternates with A.

Lemma 14. *For $k \geq 10$, if $|Z - A| > \tau$, B exists, and $|B - A| \leq 2$, $Credits(P) \geq Special(P)$.*

Proof. Let X be the largest set alternating with Z. Note that because $|X - A| \leq \tau$ by definition, $|Z - X| \geq (k-1) - \tau > 2$, so $|Z - X| \geq k - 1$. The condition on k implies $\tau > 4$. In the original variant of the lemma, we said that this subpath had $5k+2$ potential special edges: k edges incident to A, $2k$ incident to each of Y and Z, and 2 with both endpoints in $Y - A$. Consider 3 cases for the edges that are incident to A. We will show that in all three cases, we do not need to pay additional credits for the edges incident to A, either because A is saturated or because we have already counted the edges incident to A.

Case 1: A is a non-chain node. If A is 2-way critical, then A is saturated. Otherwise, without loss of generality say that A is only out-critical and thus the entering edges of B are all possible special edges incident to A. If $|B - A| = 2$, then $B = Y$ and its liabilities are already counted in the liabilities incident to Y. If $|B - A| = 1$, all the entering edges of B are paid for by conservative path payment.

Case 2: There is an alternation in the path somewhere between A and the beginning of the path A_0. Then A will be saturated.

Case 3: There is no alternation between A and A_0. In this case, Lemma 13 will apply, and A will be saturated.

These cases show we do not need to pay for any additional special edges incident to A. Therefore, we only need to pay for $4k + 2$ special edges. We consider two cases:

Case 1: $X \neq A$. If $|Z - X| \geq k$, then $|Z - A| \geq k + 1$ and $Credits(P) \geq 4(k + 1) > 4k + 2$. Otherwise $|Z - X| = k - 1$ and because $Z - X$ is bicritical, by Lemma 3(iii) there can be at most

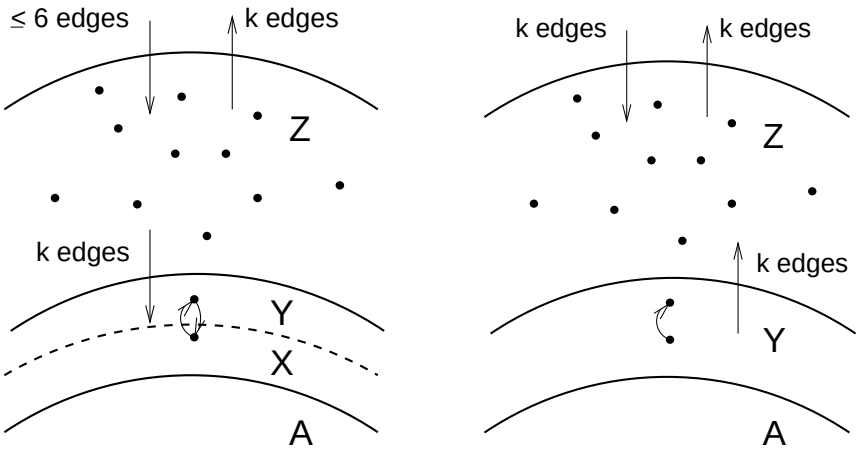
6 special edges that enter Z or leave X. In addition k special edges may enter Y, 2 special edges can have both ends in $Y - A$, and we charge Z an extra k credits in order to saturate it, giving us a total of $2k + 8$ special edges. See Fig. 4.7(a). Our total credits are:

$$Credits(P) \geq 4k = 2k + 2k > 2k + 8 = Special(P)$$

Case 2: $X = A$. In this case, all sets between A and Z are 1-way critical. Therefore, there are only k special edges leaving Y rather than the $2k$ we originally assumed, giving us only $3k + 2$ special edges we need to pay for. See Fig. 4.7(b). Our total credits are:

$$Credits(P) \geq 4(k - 1) = 3k + (k - 4) > 3k + 2 = Special(P)$$

□



(a) If $|Z - X| = k - 1$, fewer than 6 special edges enter Z.

(b) If A is the largest set that alternates with Z, all sets between A and Z are one way critical.

Figure 4.7: Proof of Lemma 14. Possible special edges for a subpath where B is the first set that alternates with A and $|B - A| \leq 2$.

Lemma 14 allows us to extend Lemma 11 to values of k between 21 and 10 under conservative path payment. However, this is insufficient to allow us to extend Lemma 12; the first set of a path

is not necessarily saturated, so we cannot assume saturation all the way through. We can show that the results of Lemma 12 still apply assuming that the path contains an alternation. We can also show a fourth result that was previously obvious but now must be proved.

Lemma 15. *Let $k \geq 10$. Consider P, the last sub path of the path $A_0, ..., A_q$. Let Z be the last set of P and A the last set before P. If the path $A_0, ..., A_q$ has an alternation, and P has any liabilities, then the following properties hold:*

(i): There is an alternation in A, P

(ii): $|Z - A| \leq 2$

(iii): Every vertex of $Z - A$ has $\geq \sigma + 1 - |Z - A|$ unused credits.

(iv): No unpaid liability of P is incident to A.

Proof. Assume A_0 is 1-way critical; otherwise, conservative path payment will saturate A_0 and Lemma 12 will apply. Let B be the first set of the path that alternates with A_0. If B is part of a subpath with more than τ vertices, then one of Lemmas 8 or 14 will saturate the last set of that subpath. If B is not part of a subpath with more than τ vertices, but $|B - A_0| > \tau$, then Lemma 13 applies and we can saturate the last set of the subpath before B. In either case, Lemma 11 applies, the set before the last subpath will be saturated, and Lemma 12 applies.

If neither of these cases apply, then B must be in P with $A_0 = A$. This gives us **(i)**, and Lemma 4 gives us **(ii)**. These in turn imply **(iii)** by the same reasoning as in the proof of Lemma 12. The only remaining task is to prove **(iv)**. If $|B - A| = 1$, conservative path payment will pay for the liabilities of B that alternate with A. Also, every liability of P which is incident to A is covered by B, so **(iv)** is proved. If $|B - A| = 2$, then $B = Z$. Let b be the number of vertices in $Z - A$ of in-degree at least $k + 1$ and d be the number of edges which go between A and $V - Z$. Then, since $B - Z$ is bicritical, by Lemma 3(ii) $b + d \leq 2$. This gives us three cases:

Case 1: $b = 2$. This implies $d = 0$, which implies no edges can enter A from outside Z.

Case 2: $b = 0$. This implies $d \leq 2$ and also no edge from A to Z can be special. The total number of special edges entering or leaving A is less than or equal to 2, which can be paid for with

our k credits.

Case 3: $b = 1$. This implies $d \leq 1$. Path payment pays for two edges between the vertices of $Z - A$, but $b = 1$ implies that no edge in $Z - A$ can be special. Therefore, we can use one of the credits that would have paid for those edges to pay for the one potential edge from outside Z to A. □

4.2.3 Liability Payment

Once these lemmas are in place, we are ready to pay for the liabilities of the children Z_i of some non-chain node W. Order the children of W so that any with no liabilities are given the lowest numbers, then those whose paths have no alternation, then finally those children with both liabilities and an alternation. For those with liabilities and an alternation, assume $|Z_i - A_i| \geq |Z_j - A_j|$ if $i > j$. The remainder of the proof is a listing of cases and a proof that in each of these cases, we have enough credits remaining to pay for our liabilities.

Let W' be the first ancestor of W that alternates with it ($W = W'$ if W is 2-way critical). Let $x = |W - \cup Z_i|$. For all Z_i, if there is an intermediate set between Z_i and A_i, call it B_i.

Define the **pass-up** as the credits beyond $2k$ in a share. For $10 \leq k < 15$, a share of credits is

$$\frac{\sigma(k+1)}{2} = 2k + 2 + \omega + \frac{\omega(\omega+1)}{10}$$

For a node at the start of a path, the first $2k$ credits will be used to satisfy its obligations under conservative path payment. The remaining $2 + \omega$ will be "passed-up" to the end of the path to help pay for any remaining liabilities. We ignore the $\frac{\omega(\omega+1)}{10}$ term.

Our cases all deal with the first two paths, $Z_0 - A_0$ and $Z_1 - A_1$. The different cases are based on the lengths of these paths and whether or not they contain alternations.

(1) At least one subpath has no alternation.

   a. Z_0's subpath has no liabilities.

   b. Neither Z_0's subpath nor Z_1's subpath contain an alternation, but both have liabilities.

c. Z_0's subpath has no alternation but Z_1's does, and both have liabilities.

(2) Both subpaths have an alternation.

a. $|Z_0 - A_0| = |Z_1 - A_1| = 1$

b. $|Z_0 - A_0| = 1$, $|Z_1 - A_1| = 2$

c. $|Z_0 - A_0| = |Z_1 - A_1| = 2$ and W has exactly 2 children.

d. $|Z_0 - A_0| = |Z_1 - A_1| = 1$ and W has 3 or more children.

In all cases except the last one, we will pay for the remaining liabilities of Z_i, $i \geq 2$ by using the bonus shares of W.

Case 1: At least one subpath has no alternation

Case 1a: Z_0's subpath has no liabilities

Say Z_0 is in-critical. Assuming that Z_1's subpath has liabilities, we are guaranteed at least

$$4x + 4 + 2(2 + \omega) = 4x + 8 + 2\omega$$

credits from the vertices of W, the vertices of Z_1, and the pass-up of Z_0 and Z_1. We will first show that we can pay for all liabilities if either $x > 0$ or W has more than 2 children. We will then consider the case where $W = Z_0 \cup Z_1$.

If $|W' - W| > 1$ or W' does not exist, conservative path payment returns the k credits. We will have $18 + 2\omega + 4x$ credits. If $x > 0$, this is enough credits to let us pay for the $20 + 2\omega$ potential liabilities of Z_1. Similarly, if W has more than 2 children, there will be 2 additional credits from Z_2's pass-up, and we will have at least $20 + 2\omega$ credits.

If $|W' - W| \leq 1$, there are $10 + \omega$ liabilities entering Z_1 and $|Z_1 - A_1|(x + 1)$ liabilities from Z_0 to the vertices of $W' - W$, and again we will have enough credits if $x > 0$ or W has more than two children. In either case, we will have enough credits unless $W = Z_0 \cup Z_1$.

Say that $W = Z_0 \cup Z_1$. We begin with the case that W is in-critical. In this case, if Z_1 is also in-critical, every entering edge covered by W is also covered by either Z_0 or Z_1. Use the k credits

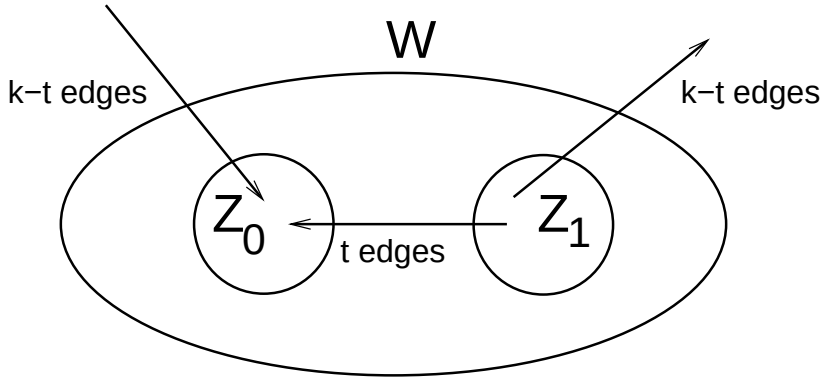


Figure 4.8: Proof of Case 1a, assuming $W = Z_0 \cup Z_1$, W and Z_0 are in-critical, and Z_1 is out-critical (and not in-critical). If t liabilities of Z_1 enter Z_0, then exactly $k - t$ edges enter Z_0 from outside W.

that pay for W's entering edges to pay for Z_1's entering liabilities instead. If Z_1 is only out-critical, then the special edges must be arranged as in Fig. 4.8. Note that any liabilities of Z_1 that enter Z_0 are already paid for. If t liabilities of Z_1 enter Z_0, then exactly $k - t$ edges enter Z_0 from outside W. Edges entering Z_0 from outside W are paid for by Z_0, so there is no need to charge W for those edges. Instead, charge W $k - t$ credits to pay for the outstanding $k - t$ liabilities of Z_1.

Finally, we deal with the case where $W = Z_0 \cup Z_1$ and W only is out-critical. In this case, we have no liabilities leaving Z_1 and at most $10+\omega$ entering it. If $|Z_1 - A_1| = 2$, we have $10+2\omega$ credits, enough to pay for entering liabilities. If $|Z_1 - A_1| = 1$ and Z_1 actually has entering liabilities, then Z_1 must be in-critical, so all three sets must be 2-way critical by Lemma 2. This contradicts our assumption that W was only out-critical, completing the case.

Case 1b: Neither Z_0's subpath nor Z_1's subpath contain an alternation, but both have liabilities

From the pass-up and the credits on the vertices of Z_0 and Z_1, we have at least $12 + 2\omega$ credits. If $|W - \bigcup Z_i| \geq 2$ or if conservative path payment returns k credits (so $|W' - W| > 1$ or W' does not exist), we have $20 + 2\omega$ credits, enough to pay for all $2k$ liabilities of Z_0 and Z_1. Thus, assume that $|W - \bigcup Z_i| \leq 1$ and $|W' - W| \leq 1$. We first consider the case where both sets are same-way critical, then the case where they are opposite-way critical.

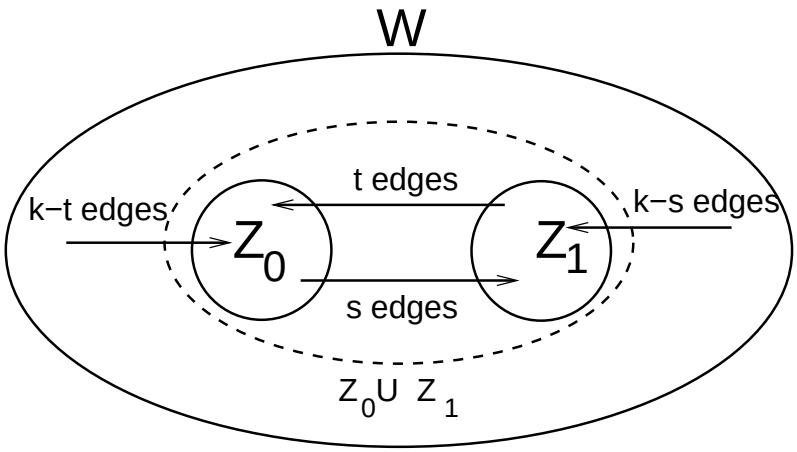


Figure 4.9: Proof of Case 1b, assuming Z_0 and Z_1 are in-critical. There are t edges from Z_1 to Z_0 and s from Z_0 to Z_1. Since $Z_0 \cup Z_1$ is a set in a k-connected graph, at least k edges enter it, which implies $t + s \leq k$.

First, consider the case where Z_0 and Z_1 (and all sets in their paths) are in-critical. Consider the set $Z_0 \cup Z_1$. This is a set of vertices in a k-edge-connected graph, so it must have in-degree at least k, which implies that there can be at most k edges which go between Z_0 and Z_1, as shown in Fig. 4.9. We can pay for these k edges between the two sets using the credits on the vertices of Z_0, Z_1, and $W - \bigcup Z_i$, leaving at least $|W' - \bigcup Z_i|$ credits on each vertex of Z_0 and Z_1, giving us enough credits to pay for the edges to the vertices of $W' - \bigcup Z_i$. A parallel case occurs if both sets are out-critical.

In the case where Z_0 is in-critical while Z_1 is out-critical, we can use k credits to pay for all of the liabilities of Z_1, again leaving at least 2 credits on each vertex of Z_0 to pay for its liabilities to the at most 2 vertices of $W' - \bigcup Z_i$. Again, a parallel case holds if Z_1 is in-critical while Z_0 is out-critical.

Case 1c: Z_0's subpath has no alternation but Z_1's does, and both have liabilities

WLOG, say Z_0 is only in-critical. We will consider two cases, a general case that will apply in almost all circumstances and a special case if $|Z_0 - A_0| + |Z_1 - A_1| \geq 5$, Z_1 is out-critical, and $W - \bigcup Z_i$ is non-empty.

For the general case, using the pass-up, the credits on the vertices of Z_1, and any credits

needed from the vertices of Z_0, pay for the k entering liabilities of Z_1. After these liabilities have been paid for, there will still be at least $|Z_1 - A_1| + 1$ credits on all vertices of Z_0 (or $|Z_1 - A_1| + 2$ if $|Z_0 - A_0| = 1$ and $|Z_1 - A_1| = 2$). If conservative path payment returns k credits (so $|W' - W| > 1$ or W' does not exist), use those to pay for either the entering liabilities of Z_0 or the leaving liabilities of Z_1, whichever alternates with W. If $|W' - W| \leq 1$, pay for the liabilities between the possible vertex of $W' - W$ and either Z_0 or Z_1 using the credits on the vertices of Z_0, leaving $|Z_1 - A_1|$ credits on each vertex of Z_0. Pay for the liabilities from Z_0 to Z_1 using the credits on the vertices of Z_0. Pay for any liabilities between the vertices of $Z_i - A_i$ and each vertex of $|W - \bigcup Z_i|$ using the at least 4 credits on each vertex of $W - \bigcup Z_i$.

The above scheme will allow us to pay for all liabilities except in the case that $|Z_0 - A_0| + |Z_1 - A_1| \geq 5$, Z_1 is out-critical, and $W - \bigcup Z_i$ is non-empty. If $|Z_0 - A_0| = 4$, then there are $10 + \frac{4\omega}{5}$ credits on the vertices of Z_0. Use these along with $\frac{\omega}{5}$ credits from the vertices of Z_1 to pay for the liabilities of Z_0 and pay for the liabilities of Z_1 as described in Case 1a. Otherwise, if $|Z_0 - A_0| + |Z_1 - A_1| \geq 5$ but $|Z_0 - A_0| < 4$, it must be the case that $|Z_0 - A_0| = 3$ and $|Z_1 - A_1| = 2$. In this case, there may be 5 liabilities involving each vertex of $W - \bigcup Z_i$, and we only have 4 credits to pay for them. However, we have $23 + 2\omega$ credits from the pass-up and the credits on Z_0, Z_1, and the first vertex of $W - \bigcup Z_i$. This gives us enough credits to pay for the entering liabilities of Z_1 and either the entering liabilities of Z_0 or the leaving liabilities of Z_1, depending on which alternates with W, leaving 3 credits left over, which will be enough to pay for any remaining liabilities to the first vertex of $W - \bigcup Z_i$. There will be at most 3 liabilities to the remaining vertices of $W - \bigcup Z_i$, which can be paid for by the credits on those vertices.

Case 2: Both sets have an alternation

For the cases where both sets have an alternation, assume that W is out-critical. Also note that if $x \geq 7$, then there will be enough credits on the vertices of W to pay for all liabilities of Z_0 and Z_1, so assume that $x \leq 6$.

In all cases, note that if $|W' - W| > 1$ or if W' does not exist, conservative path payment

returns k credits. Therefore, if conservative path payment does not return k-credits, this implies that W' exists. Because W' is a critical subset, $W' \neq V$.

Case 2a: $|Z_0 - A_0| = |Z_1 - A_1| = 1$

If $|Z_0 - A_0| = |Z_1 - A_1| = 1$, then we will have $4x + 12 + 2\omega$ credits on the vertices and from the the pass-up. We will consider the cases where $|W' - W| \leq 1$ and where conservative path payment returns k credits.

In the case that $|W' - W| \leq 1$, then we will have $4x + 2$ liabilities to the vertices of $W' - \bigcup Z_i$ and 2 between Z_0 and Z_1 for a total of $4x + 4$ liabilities, less than our $4x + 12$ credits.

In the case that conservative path payment returns k credits, we will have $20 + 2\omega$ liabilities entering Z_0 and Z_1, 2 between the vertices of Z_0 and Z_1, and $2x$ to the vertices of $W - \bigcup Z_i$. We will have $4x + 22 + 3\omega$ credits, enough to pay for our $2x + 22 + 2\omega$ liabilities.

Case 2b: $|Z_0 - A_0| = 1$**,** $|Z_1 - A_1| = 2$

In this case, we have $4x + 14$ credits from the vertices and the pass-up. We will consider the case where conservative path payment returns k-credits, then several subcases if $|W' - W| \leq 1$.

If conservative path payment returns k credits, then there are $4x + 24 + 3\omega$ credits, which is less than the $3x + 4 + 20 + 2\omega$ potential liabilities.

In the case that $|W' - W| \leq 1$ and at least one of Z_0 or Z_1 is 1-way critical, then our total liabilities are $5x + 4 + 3$ which is less than our $4x + 14$ credits when $x \leq 7$.

In the case that $|W' - W| \leq 1$ and there are at least three children of W, use the pass-up from Z_2 to obtain a total of $4x + 18 + 4\omega$ credits. If $x \leq 5$, this is less than our potential $6x + 7$ liabilities. If $x \geq 6$, then we have at least $24 + 18 + 4\omega > 4k$ credits, enough to cover all liabilities.

If none of the above cases hold, then $|W' - W| \leq 1$, there are only two children, and both Z_0 and Z_1 are 2-way critical. Consider the case where W is 2-way critical. We have $6x + 4$ liabilities, less than our credits if $x \leq 5$. Because Z_0 and Z_1 are both 2-way critical, $W - Z_0 - Z_1$ is tricritical, so by Lemma 5 $x \neq 5$ or 6. Next, consider the case where $|W' - W| = 1$. We may have $6x + 4 + 3$ liabilities, less than our credits when $x \leq 3$. If $W' \neq W$, the fact that Z_0 and Z_1 are both 2-way

critical implies that both $W - Z_0 - Z_1$ and $W' - Z_0 - Z_1$ are tricritical. Thus, Lemma 5 implies $x \neq 4$, 5, or 6. Therefore, either, $x \leq 3$ or $x \geq 7$, and in either case, we can pay for all liabilities.

Case 2c: $|Z_0 - A_0| = |Z_1 - A_1| = 2$ and W has exactly 2 children

Again, we will have separate cases if conservative path payment returns k credits or if $|W' - W| \leq 1$. In the case that $|W' - W| \leq 1$, we will either have a lemma that allows us to pay for all liabilities or we will be guaranteed a tricritical set. Further subcases will depend on exactly what this tricritical set is.

In the case that conservative path payment returns k credits, then we have $4x + 26 + 3\omega$ credits. We have at most $20 + 2\omega$ entering liabilities. Our leaving liabilities depend on the first in-critical set of Z_1. If Z_1 or B_1 is in-critical, then at least 2 of the 8 potential edges between Z_0 and Z_1 are entering liabilities of Z_1. Thus, we have only $6 + 4x$ leaving liabilities and $4x + 26 + 2\omega$ total liabilities. If the first in-critical set of Z_1 is A_1, then Z_1 didn't actually have any entering liabilities and we only have $4x + 18 + 2\omega$ total liabilities. In either case, we can pay for all liabilities.

If $|W' - W| \leq 1$, recall that W' exists, W' is a critical subset, and therefore $W' \neq V$. We need the following lemma:

Lemma 16. *If W has exactly two children and $|Z_i - A_i| = 2$ for all $i \in 0, 1$, then we can pay for all edges if any of the following hold:*

a) $x \geq 6$

b) There are no more than 2 in-special vertices (vertices with in-degree greater than k) in $W' - Z_0 - Z_1$.

c) At least one of the Z_i is 1-way critical and does not have a B_i which alternates with it.

Proof. For part (a), observe that we have

$$(4 + \frac{\omega}{5})x + (12 + \frac{4\omega}{5}) + (4 + 2\omega) \geq 40 + 4\omega$$

credits when $x \geq 6$. For part (b), let there be t special vertices. Then our liabilities will be $8 + 4x + 4t$, less than our $4x + 16$ credits when $t \leq 2$. For part (c), say Z_0 is our 1-way critical

set, and B_0, if it exists, does not alternate with Z_0. Use k credits to pay for the liabilities of Z_0, leaving at least $4x + 6$ to pay for the liabilities of Z_1. There are potentially 4 remaining liabilities between the vertices of Z_0 and Z_1. There will be at most 2 liabilities between Z_1 and $W' - W$. There could be as many as $4x$ liabilities between Z_1 and $W - Z_0 - Z_1$. This gives us a maximum of $4x + 6$ liabilities, which we can pay for with our remaining credits. □

If the condition in (c) does not hold, then for $i \in \{0, 1\}$, either B_i or Z_i alternates with W'. Thus, we are guaranteed that one of the following sets (illustrated in Fig. 4.10) is tricritical:

(1) $W' - Z_0 - Z_1$

(2) $W' - Z_0 - B_1$

(3) $W' - B_0 - Z_1$

(4) $W' - B_0 - B_1$

If (2) is tricritical, then either part (b) of Lemma 16 holds or we are guaranteed that that the tricritical set has at least 3 special vertices and 4 total vertices, because we have 1 vertex in $Z_1 - B_1$ and at least the 3 in-special vertices in $W' - Z_0 - Z_1$. Therefore, by Lemma 6 it must be larger than 8, giving us $x + 1 + 1 \geq 8$, so $x \geq 6$. A parallel argument holds for (3).

Say that (4) is tricritical, but (2) and (3) are not. If part (b) of Lemma 16 does not hold, (4) has more than 3 vertices, at least 3 of which are special, so Lemma 6 implies $x + 2 + 1 \geq 8$, or $x \geq 5$. Also, in this case, we are guaranteed that both Z_i are in-critical, so $W - Z_0 - Z_1$ is tricritical. Thus $x \neq 5$ or 6, and so $x \geq 7$.

Say that (1) is tricritical, but none of the other three are. In this case, we are guaranteed that both Z_i are out-critical (possibly 2-way critical), and the B_i, if they exist, are only in-critical. If $W' = W$, then if none of the other sets are tricritical, neither B_i can exist. In this case, we have $20 + 2\omega = 2k$ credits on the vertices of Z_i and from the pass-up. Use these to pay for the leaving liabilities of both Z_i. There may be an additional $4x$ liabilities to the vertices of $W - \bigcup Z_i$, which can be paid for using the credits on the vertices of $W - \bigcup Z_i$.

Otherwise, $|W' - W| = 1$. We know that (1) has cardinality $x + 1$. If part (b) of Lemma 16 does not hold, the set has at least 3 in-special vertices, and thus $x + 1 \geq 8$ or $x + 1 = 3$, implying $x \geq 7$ or $x = 2$. If $x \geq 7$, part (a) of Lemma 16 holds, so assume $x = 2$.

If $x = 2$ and neither B_i exists, we have $4x + 20$ credits and $8x + 4 + 8$ liabilities. Our credits are less than our liabilities when $x \leq 2$.

If at least one B_i, say B_0, exists, then it is not possible that $x = 2$. If A_1 is out-critical, $W' - Z_0 - A_1$ is tricritical. If A_1 is in-critical, $W - B_0 - A_1$ is tricritical. Whether A_1 is out-critical or in-critical, we have a tricritical set of cardinality $x + 3$. Thus, by Lemma 5, $x \neq 2$ or 3.

Case 2d: $|Z_0 - A_0| = |Z_1 - A_1| = 2$ and W has 3 or more children

In this case, we will want to pay for the liabilities of Z_0, Z_1, and Z_2 using the credits on the vertices, the pass-up, and the bonus share of Z_2. For any Z_i, $i > 2$, we will make use of the following lemma:

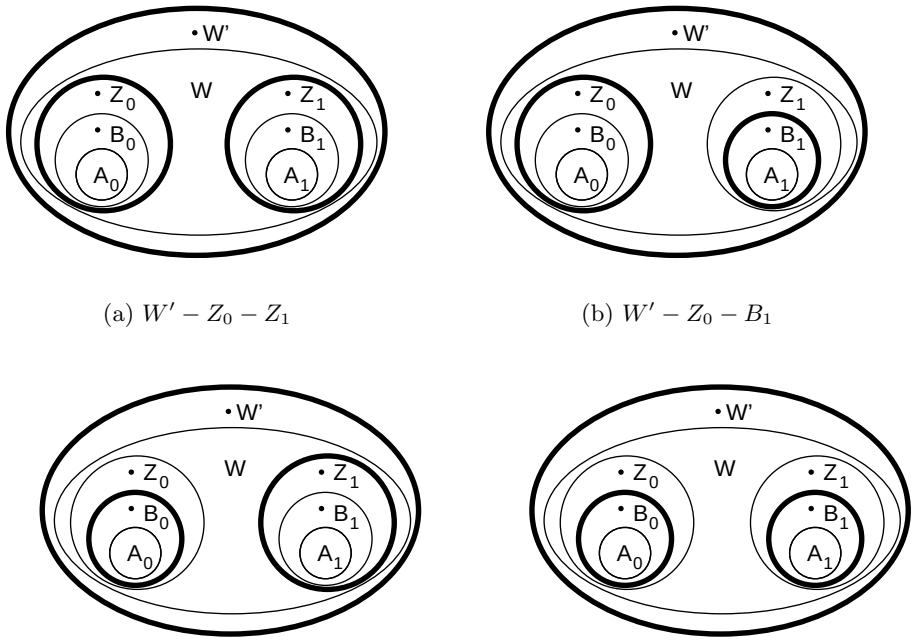
Lemma 17. *Suppose that $|Z_i - A_i| = 2$ for all children of a non-chain node W. Consider a child Z_i for some $i > 2$. Using only Z_i's bonus share and the credits on its vertices, we can pay for all liabilities of Z_i and any edges from Z_0, Z_1, or Z_2 to Z_i.*

Proof. Use $2k$ credits from the bonus share to pay for entering and leaving liabilities of Z_i. In order to prove that there are enough credits to pay for edges from the first three subpaths, look at the last in-critical set of Z_i's subpath.

If Z_i is the last in-critical set, then all edges that enter Z_i are liabilities of Z_i and were already paid for.

If B_i is the last in-critical set, there may be 6 unpaid edges from Z_0, Z_1, and Z_2 to $|Z_i - B_i|$. These can be paid for using the 6 credits on the vertices of $|Z_i - B_i|$.

If A_i is the first in-critical set, then the path didn't actually have any entering liabilities. Therefore, k credits from the bonus share can be used to pay for additional edges. This gives us at least 16 credits, more than enough to pay for the 12 possible entering edges from Z_0, Z_1, and Z_2. □



(a) $W' - Z_0 - Z_1$ (b) $W' - Z_0 - B_1$

(c) $W' - B_0 - Z_1$ (d) $W' - B_0 - B_1$

Figure 4.10: Tricricial sets in the proof of Case 2c. If condition (c) in Lemma 16 does not hold, one of the sets shown must be tricritical.

This lemma insures that we do not need to worry about any leaving liabilities of Z_0, Z_1, and Z_2 that may be incident to any other Z_i, so we will be able to ignore the Z_i, $i > 2$ for the rest of the proof.

For Z_0, Z_1, and Z_2, we will first try to pay for the $3k$ entering liabilities of all sets, then pay for the edges between the three of them and to the vertices of $W - \bigcup Z_i$. Note that we have

$$2k + 4(2 + \omega) + 18 + 4x = 4k + 6 + 2\omega + 4x$$

credits. When $x \geq 4$ this is at least $4k + 22 + 2\omega$, enough to pay for all $6k$ potential liabilities, so we can assume $x \leq 3$. We will have 4 cases, depending on where each subpath has its largest in-critical set.

Case 1: A_i is the largest in-critical set for at least 2 subpaths. In this case, there are at most $4k$ entering and leaving liabilities over all 3 subpaths and at least $4k + 6$ credits.

Case 2: If A_i is the largest in-critical set for exactly 1 subpath. After paying for the $2k$ entering liabilities of the other 2 subpaths, we will have at least $26 + 4x$ credits remaining. In this case, there may be 16 liabilities between the vertices of Z_0, Z_1, and Z_2 and $6x$ to the vertices of $W - \bigcup Z_i$. We have enough credits to pay for all liabilities when $x \leq 5$.

Case 3: None of the subpaths have A_i as their largest in-critical set, but at least one, say Z_0, has Z_i as its largest in-critical set. Each vertex of Z_0 may have 2 leaving liabilities to the other subpaths, and each vertex of Z_1 and Z_2 may have 1 leaving liability to the other subpaths. This gives us a total of 8 liabilities between the subpaths and $6x$ to the vertices of $W - \bigcup Z_i$. After paying for the $3k$ entering liabilities, we will have at least $16 + 4x$ credits, which is enough to cover our liabilities when $x \leq 4$.

Case 4: All 3 subpaths have B_i as their largest in-critical set. In this case, there may be 12 liabilities between the subpaths and $6x$ to the vertices of $W - \bigcup Z_i$. As in the previous case, we have at least $16 + 4x$ credits after paying for all entering liabilities, which may be 2 less than our remaining liabilities when $x = 3$. However, recall from the proof of Lemma 10 that the reason that we have 3 and not 4 credits on the vertices is that we already paid for an edge between the vertex

of $|Z_i - B_i|$ and the vertex of $|B_i - A_i|$. Assuming that this edge exists, it is one of the k entering liabilities of B_i. Because it was already paid for, we only need to pay for $k-1$ additional edges. If this edge doesn't exist, then we have an additional credit on the vertices of $|Z_i - A_i|$. Either way, we will have 3 additional credits, enough to cover all liabilities.

Chapter 5

An introduction to computational biology and protein-protein interaction networks

Computers and computer science are acquiring an ever more important role in modern biology. With new lab techniques, biologists are able to generate an exponentially increasing amount of data. This data must be processed and analyzed if it is to be used in further biological research. The need to process this data has spurred on the field of computational biology, the science of developing new algorithms for biological data and finding ways to apply existing algorithms to new biological problems.

Many types of biological data can be represented as graphs making graph theory one of the subfields of computer science with significant applications to computational biology. Some of these graphs of biological data, usually called biological networks, include metabolic networks that model the various chemical reactions in the cell, genetic co-expression networks that have edges between genes that are expressed together, and protein-protein interaction networks that contain information about which proteins bind to each other.

5.1 Protein-Protein Interaction Networks

Protein-protein interaction (PPI) networks are graphs designed to represent protein interactions in the cell. Vertices represent proteins, and there is an edge between two vertices if the corresponding proteins have been shown to interact physically (or sometimes just predicted to interact) in some study [22]. Examples are shown in Fig. 5.1 and 5.2. Several different types of

assays determine interactions. Much of the interaction data comprising PPI networks are generated from high throughput assays which identify many interactions in a relatively short period of time. Two common high throughput methods are **yeast 2-hybrid (Y2H)** and **affinity purification** [23]. Y2H assays use tagged proteins to determine binary interactions. The tags on the proteins are part of the GAL4 protein from the yeast *Saccharomyces cerevisiae*, a protein involved in regulating transcription. The protein is divided into two parts, an activation domain and a DNA binding domain. When the two parts are brought into close proximity, they activate the transcription of a reporter gene [24]. See Fig 5.3. Affinity purification also uses a tagged protein to determine interactions. A particular protein, the "bait," is tagged in such a way that it will make it easy to remove from the cell, and that the tag does not interfere with the natural level of protein expression in the cell. Affinity purification removes this bait protein from the cell along with anything attached to the bait. A subsequent assay, usually mass spectrometry, determines which proteins have been brought out [25]. See Fig. 5.4. However, mass spectrometry only determines which proteins are in the sample, not the binary interactions within it. Therefore, interactions from affinity purification can either be modeled using a "hub-and-spoke" pattern where there is an edge between the bait protein and each protein pulled out with it, or a clique where there is an edge between each protein pulled out [26]. See Fig. 5.5.

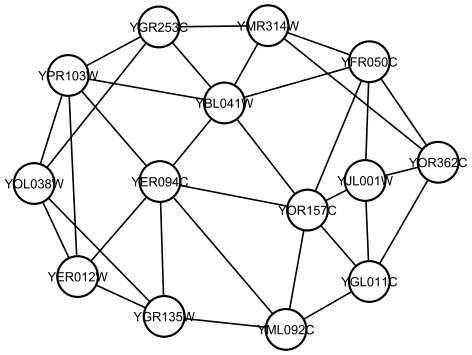


Figure 5.1: A small example of a protein-protein interaction graph.

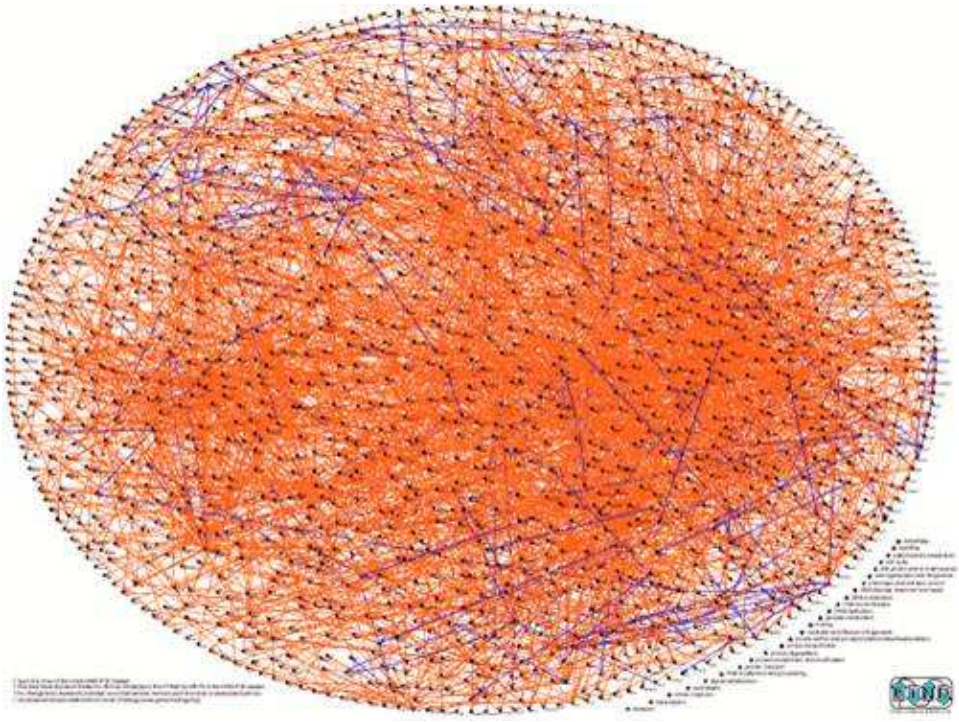


Figure 5.2: An example of a protein-protein interaction network. Image is from "Yeast Proteomics", Genome News Network, 1-18-02 [27].

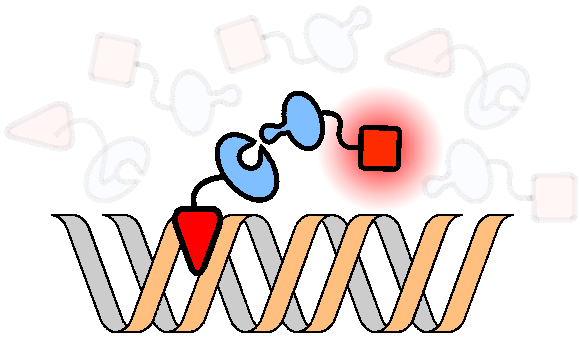


Figure 5.3: An example of yeast 2-hybrid. Blue shapes are proteins. Red shapes are tags. When the two proteins with different tags bind to each other, the tags are brought close enough together to initiate transcription of a reporter gene that indicates an interaction has occurred. Image is from Gibson and Goldberg [28].

Both types of assays to determine the edges are error-prone, and have both false positives (proteins that don't interact but appear to) and false negatives (proteins that do interact, but whose interaction has not been captured). Nonetheless, protein-protein interaction networks are

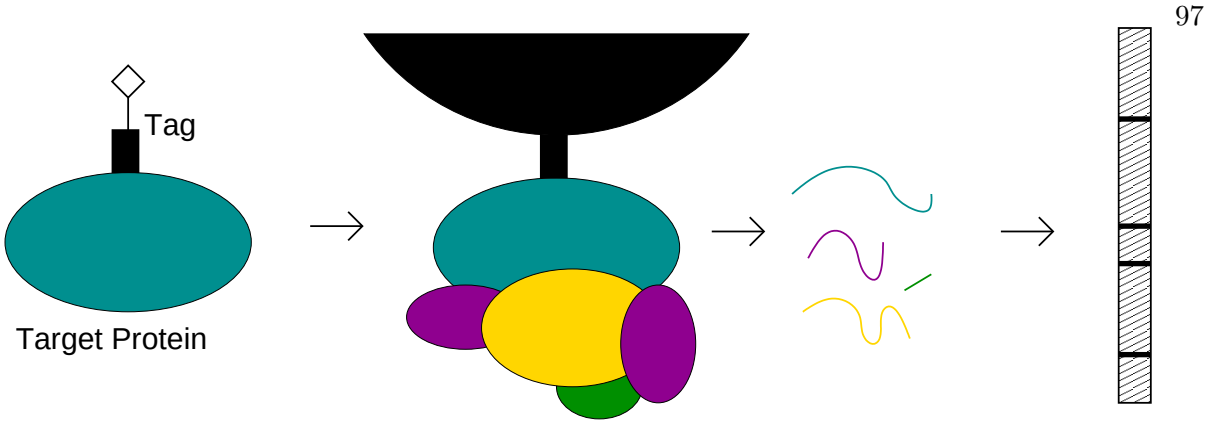


Figure 5.4: An example of affinity purification. The blue protein is tagged to allow it to be removed from the cell. When it is removed, the magenta, yellow, and green proteins are removed with it. The resulting combination of proteins is disassembled and run through mass spectrometry to determine which proteins were pulled out.

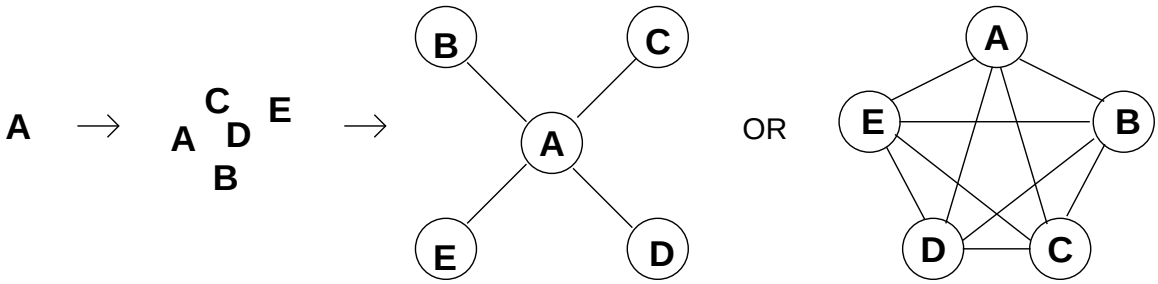


Figure 5.5: An example of affinity purification. Protein A is the bait. When it is pulled out, proteins B, C, D, and E are brought out with it. In the PPI network, this is modeled as either the "hub-and-spoke" graph on the left or the 5-clique on the right.

valuable tools for studying the proteome of an organism.

5.1.1 Topology of PPI networks

There are many reasons to study PPI network topology. We can compare the topology of PPI networks to the topology of other, well-studied networks. This comparison will demonstrate in what ways PPI networks are similar to these other networks, and therefore which knowledge about these networks can be used in our study of the PPI network. Patterns in the edges may provide evidence to discern the difference between edges that represent true interactions and those that are

false positives and allow us to assign an improved confidence score to individual edges. Looking at edge topology can also help determine the nature of the interactions those edges represent; are these interactions long-term interactions, such as those that are part of a complex, or transitory interactions that occur only briefly under special circumstances. Studying PPI network topology also allows scientists to evaluate existing models of how protein interaction works and create better models. Understanding the topology of the PPI network can give a better understanding of the underlying evolutionary mechanisms that created that network.

Due to the fact that network theory is a relatively new field, there are very few analytic methods for determining the significance of a network's properties. Therefore, in order to determine which topological properties are significant in a PPI network, the properties are usually compared against those in random networks. The question is, what is the appropriate way to create random networks for comparison? The number of vertices and edges are usually held constant so that we can determine which properties are significant and which are expected given the number of vertices and edges in the graph, but determining the appropriate distribution of edges is more difficult. Traditionally, the random model used for graphs is the Erdős-Rényi random graph [29]. Erdős-Rényi graphs have n vertices, and between any two vertices, there is an edge with probability p. By setting n equal to the number of vertices in the PPI network and p equal to the edge density, a graph with the same number of vertices and roughly the same number of edges is obtained. The properties of this random graph can be compared with those of the PPI network in order to determine the significance of these properties.

Because the properties of Erdős-Rényi graphs differ significantly from those of PPI networks, it is now considered to be an inappropriate random model for determining the significance of PPI network properties. Several other random models have been developed. One common method is to generate a random network with the same number of vertices of each degree as the target network. Another method is to hold the network itself constant and randomly reassign labels (protein or gene names) to the vertices; this method is effective if we want to study the properties vertices connected by an edge (i.e., whether or not two proteins with similar function are more likely to

interact), but it is useless if the topological features of the network as a whole are being assessed.

There are many topological properties often studied in connection with PPI networks. Some of the most common are listed below:

Edge Density: For the network or a subgraph in the network, the number of actual interactions divided by the possible interactions.

Degree Statistics: The maximum and mean degrees in the network.

Degree Distribution: How many vertices of each degree are present in a network.

Clustering Coefficient (CC): A measure of how many of a vertex's neighbors are neighbors of each other. For a single vertex, the clustering coefficient is the edge density among the vertex's neighbors. We also want to look at clustering coefficient over a graph or subgraph. Clustering coefficient over a graph was originally defined as the average of the clustering coefficient of the vertices, and this definition is still used occasionally. However, this definition is problematic when the network contains degree 1 vertices, which have undefined clustering coefficients. This definition also overemphasizes low-degree vertices. Therefore, an alternate definition is usually used: clustering coefficient in a graph is defined as 3 times the number of triangles divided by the number of length 2 paths. See Fig. 5.6.

Mutual Clustering Coefficient (MCC): For a pair of vertices, a measure of how many neighbors they share. This is often a ratio; the top number in the ratio is always the number of shared neighbors, but there are several possible denominators including the size of the union of their neighborhoods, the size of the minimum neighborhood, or some way of combining the two. An alternative to the ratio method looks at the overlap and calculates the probability that this overlap is due to chance. These various methods are evaluated in Goldberg and Roth [30].

Motifs: Particular subgraphs, such as cycles or cliques of a particular size, in the network. Depending on the application, motifs can either be arbitrary subgraphs or may be forced to be induced subgraphs. There are two connected motifs of size 3 (a path and a triangle), six connected motifs of size 4 (a path, a Y-shape, a 4-cycle, a 4-cycle with a chord, a triangle with an additional vertex, and a 4-clique), twenty-one of size five, etc.

Path length statistics: The maximum and mean length (called the characteristic path length) of the shortest path between any two vertices in the network.

Betweenness Centrality: Betweenness centrality is defined on either a vertex or an edge and is the number of shortest paths of which that vertex or edge is a part. To calculate betweenness centrality, all shortest paths between all pairs of vertices should be calculated. Then, determine how many of those shortest paths the desired vertex or edge is on. That number is the betweenness centrality of the vertex or edge. Betweenness centrality is often used to divide the graph; edges or vertices of high betweenness are removed in order to divide the graph into multiple connected components.

k**-core**: A subgraph of the network where every vertex in the subgraph has degree at least k within the subgraph. Note that this is a weaker condition than k-edge or k-vertex-connectivity; a k-core is not necessarily k-edge- or k-vertex-connected, but a k-connected graph must be a k-core.

There are many topological features common to protein-protein interaction networks across species. PPI networks tend to have high clustering coefficients when compared to equivalent random networks; an edge between two vertices a and b and another edge between b and c greatly increases the probability that there is an edge between a and c [31]. Despite this high clustering, however, PPI networks tend to have short average path lengths; although the network is large, there tends to be a short path between any two vertices in the network [32, 33]. Graphs that have both of these properties, high clustering coefficients and short average path lengths, are often referred to as **small-world** networks [34]. PPI networks are also said to have a power-law degree distribution: rather than the degrees of nodes having a Poisson distribution like Erdős-Rényi random graphs [29], there are a small but significant number of very high degree nodes [35, 36]. This type of network is usually referred to as a **scale-free** network. See Fig. 5.7.

5.2 Studying PPI networks

Protein interaction network research can help determine the function of proteins, divide the network into functional modules, and find protein complexes. These goals are closely related, and

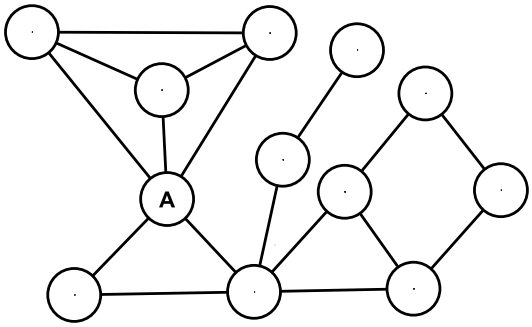
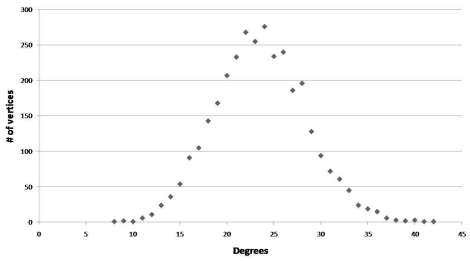
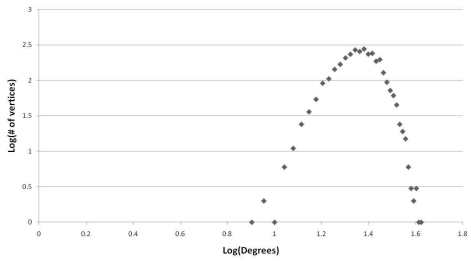


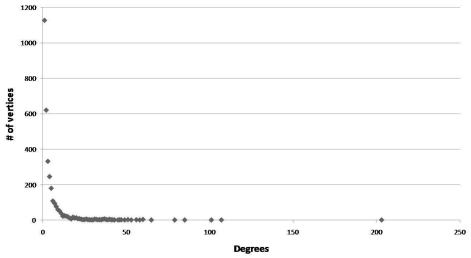
Figure 5.6: Clustering coefficients on vertices and graphs. Vertex A has a clustering coefficient of 0.4 because 4 of the 10 possible interactions between its 5 neighbors are present. The entire graph has a clustering coefficient of 0.46 because it has 6 triangles and 39 length 2 paths.



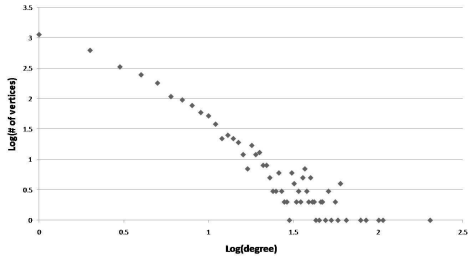
(a) Degree distribution from an Erdős-Rényi random graph.



(b) Log-log plot of the degree distribution in (a).



(c) Degree distribution from the Yeast Y2H interaction network.



(d) Log-log plot of the degree distribution in (c). Note that it is roughly linear.

Figure 5.7: The difference in degree distributions between a random graph and a PPI network. While the random graph has a bell-curve distribution around the average degree, the PPI network has many low degree nodes and a small but significant number with very high degrees.

many of the same techniques are used for more than one of these tasks.

5.2.1 Predicting protein function

One common goal of studying the PPI network is to predict the function of proteins which have not been completely characterized yet. Proteins are more likely to interact with other proteins of similar function, so the function of a target protein can be predicted by examining the pattern of functions of proteins in its surrounding neighborhoods.

There are many published methods to predict the functions of unknown proteins. Some of these are as simple as counting the neighbors of a target protein and assigning the most common function of the neighbors to the target [37, 38, 39], assuming that proteins are more likely to interact with those with the same function. This method is often called **guilt by association**. Other methods involve coming up with heuristics to approximate the solutions to NP-complete graph theory problems such as the multiway cut [40, 41, 42], under the assumption that proteins with similar functions are likely to have more edges between them than they do to proteins of other functions. Still others use probability and Markov random fields [43, 44], again assuming the proteins are more likely to interact with those of similar function. See Sharan et al. [45] for a more complete survey of the methods used to determine protein function based on interaction data.

5.2.2 Determining functional modules

Functional modules are subgraphs of the PPI network whose proteins are linked by common biological traits. Functional modules may be proteins with the same function, proteins that are all part of the same signaling network, or proteins that are co-located in the cell. In order for our algorithms to be able to distinguish them, these subgraphs need to be relatively isolated from the rest of the network, but it is widely believed that they will be, given the theory that proteins interact densely within the same functional group and only minimally with proteins of other function. Many biological functions occur with the interaction of multiple molecules, and learning the function of individual proteins may not be as useful as learning the function of a module [46].

One example of a functional module is a protein complex. Methods for finding protein complexes are discussed below. More information on finding other types of functional modules can be found in the review papers by Sharan et al. [45] and Qi and Ge [47].

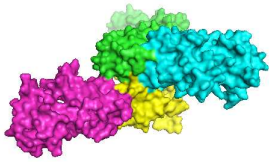
5.2.3 Finding protein complexes

Rather than performing their function alone, many proteins form **protein complexes**, groups of proteins that bind together to perform a specific task. Examples of protein complexes include the proteasome, a complex for breaking down proteins, and the DASH complex, which stabilizes the spindle during cell division. In addition to complexes such as the proteasome and the DASH which are composed entirely of proteins, there are also complexes which include both proteins and RNA. Examples of these include the ribosome, responsible for building proteins from RNA, and the various snRNPs (small nuclear ribonucleoproteins, pronounced "snurps") which are responsible for splicing the RNA prior to protein synthesis. See Fig. 5.8. Some of these complexes, such as the proteasome, are well-characterized, but others are not, and scientists believe that there are many protein complexes in the cell that have not yet been identified. Complexes play an important role in the function of the cell, and by discovering new complexes and learning more about their structure, we can gain insights into cellular biology.

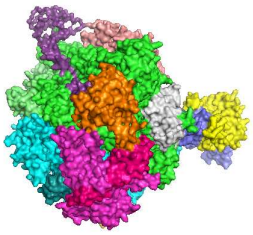
There have been many previous attempts to find protein complexes using many different techniques. Complexes are functional modules, so the methods used to find complexes are a subset of the methods discussed in Section 5.2.2. Determining whether or not a given protein is part of a complex also helps determine that protein's function, so these methods are applicable to that task as well. We discuss a few of the more prominent methods here.

5.2.3.1 MCODE

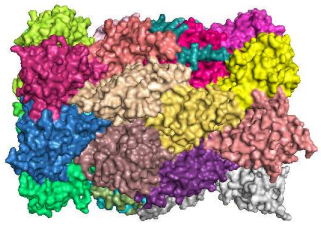
One of the best known algorithms for predicting complexes in the PPI network is the MCODE algorithm of Bader and Hogue [49]. MCODE combines degree statistics, clustering coefficient, and edge density to find subgraphs that may be complexes.



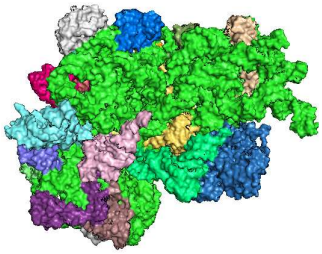
(a) The ESCRTII complex, involved in protein sorting (made from PDB code 1w7p).



(b) An RNA polymerase II elongation complex (made from PDB code 1y1v).



(c) The 20S proteasome, involved in breaking down proteins (made from PDB code 1jd2).



(d) The ribosome, a complex involved in building proteins. Unlike the other complexes here, it contains both proteins and RNA (made from PDB code 3fih).

Figure 5.8: Examples of protein and protein-RNA complexes. The ESCRTII, RNA polymerase elongation complex, and proteasome are pure protein complexes, while the ribosome contains both proteins and RNA. These are surface pictures, showing the surfaces of all the proteins involved, but not their atoms or individual structure. Structure information for images is from the protein data bank, and images made using PyMOL [48].

MCODE works in three stages: vertex weighting, complex prediction, and post-processing. In the vertex weighting stage, the algorithm looks at the neighborhood of each vertex. Within the neighborhood, the algorithm looks for the maximum k-core, a graph where every vertex has degree at least k, for the highest possible value of k. The edge density of this k-core multiplied by k is the weight of the vertex. Note that this is similar to the clustering coefficient of the vertex, except that MCODE only considers the clustering among the neighbors in the k-core rather than all neighbors.

In the complex prediction stage, the algorithm starts with the highest-weighted vertex not yet assigned to a complex as a seed. The algorithm moves out from the seed vertex, adding vertices to the complex if their weights are at least a certain fraction of the weight of the seed vertex. This fraction is an input parameter to the algorithm.

In the post-processing stage, complexes that do not contain at least a 2-core are eliminated. There are also two optional operations that may occur in the post-processing stage. The "fluff" operation adds neighboring vertices to the complex if the vertex's neighborhood has an edge density above an input threshold. The "haircut" operation recursively eliminates all vertices of degree 1, leaving a 2-core.

Although the MCODE algorithm dates back to 2003, it still has an influence on the development of other complex-finding algorithms. Software for running MCODE is still actively maintained and updated. Algorithms developed in 2010 still compare their performance to MCODE.

5.2.3.2 Edge density methods

Edge density-based algorithms are some of the most common algorithms for predicting protein complexes. It is widely believed that complexes will have many edges within the complex and relatively few edges outside the complex. A common theory takes this a step further and asserts that if all real interactions were present, a complex would appear in the data as a **clique**, a subset of vertices where all possible interactions are present within the subset; therefore, complexes will appear as subgraphs with high edge density.

Most algorithms based on edge density try to find subgraphs above a certain density threshold.

The algorithm of Spirin and Mirny, for example, starts by searching for all maximal cliques (while this problem is NP Hard, it is feasible in a sparse graph). They then look for additional complexes by trying to find a set of nodes with maximum edge density using a monte carlo algorithm [50].

Other methods try to include as many cliques as possible. The CFinder algorithm of Adamcsek et al. looks for all k-cliques for a given value of k (usually 4-6). They define adjacent k-cliques as k-cliques that share $k-1$ vertices. The potential complexes returned by CFinder are what they call k-clique percolations; two vertices are in the same k-clique percolation if there is a path between them going through adjacent k-cliques [51].

Cui et al. have a similar method to CFinder, but they consider the idea of adjacent cliques to be too strict. Rather than adjacent cliques, their algorithm returns what they call "near-cliques." They give three possible near-cliques: first, a clique and a vertex outside the clique where the vertex has at least two neighbors in the clique; second, two cliques that share at least one protein; finally, two cliques and a vertex outside those cliques such that the vertex has at least two neighbors in each of the cliques [52].

Bu et al. are also looking for subgraphs with high edge density, though they use a different method. They find "quasi-cliques" by looking for eigenvectors of the adjacency matrix with positive eigenvalues (negative eigenvalues indicate "quasi-bipartite" subgraphs, which are not believed to correspond to complexes). These quasi-cliques are returned as potential complexes [53].

King et al. use edge density as a filtering method. They first use a randomized algorithm on the graph to find a low-cost partition of the vertex sets, where the cost is related to the number of edges between elements of the partition and the number of edges not present within an element. Then, they look at the elements of the partition to determine whether or not those elements might be complexes. An element of the partition is considered a complex if it has a high enough edge density (usually between 0.65 and 0.75) and enough vertices (the cutoff point varied with the size of the network) [54].

Zotenko et al. use edge density in their algorithm to find and separate overlapping complexes. They first modify the graph by adding an edge between any two vertices that share all their

neighbors, as well as adding one edge across all 4-cycles. Then, if the graph is a chordal graph (one for which any cycle of length greater than 3 contains a chord), they build the "clique tree" representation which shows the maximal cliques and their relationship to each other. The maximal cliques are predicted to be functional groups or complexes in the interaction network [55].

5.2.3.3 Betweenness methods

Most methods for finding complexes try to find the edges and vertices that are most central to a complex, but a few methods turn this around. Rather than trying to find the edges and vertices that are most central and group those together into complexes, they try to find those that are least central and remove them under the theory that what remains will give us a much better idea of the structure of complexes in the graph. The methods that work under this theory usually use the property of betweenness described in Section 5.1.1 as a method for finding less central edges. The theory for the betweenness methods is the same as the theory for edge density, that complexes will have many edges within the complex and relatively few edges outside the complex, but betweenness methods look for the edges outside rather than those within. According to this theory, any shortest path between two vertices in different complexes will be forced to go through a small number of edges, so these edges between complexes will have high betweenness.

The first attempt to apply this theory to biological networks was Girvan and Newman's algorithm for hierarchical clustering [56]. They calculated the betweenness centrality of all edges in the graph, removed the edge with the highest betweenness, then recalculated the betweenness after removing the edge. This procedure was repeated until there were no more edges. The result is a tree where closely related vertices are grouped together.

Since Girvan and Newman introduced betweenness centrality to biological networks, there have been several attempts to apply betweenness to the protein-protein interaction network. Chen and Yuan run a similar algorithm to Girvan and Newman's on the PPI network to find complexes and other modules. There were two major modifications that Chen and Yuan made to the Girvan-Newman method. First, they weighted each edge of the PPI network based on the "dissimilarity"

of the expression of the genes that create the proteins represented by each endpoint. These weights were taken into account when calculating the shortest paths to determine betweenness. Chen and Yuan also removed "redundant" paths from their calculation that contained the same start or end vertices; for a given edge, if the calculation of its betweenness used the shortest path between vertices u and v, the calculation could not include any other shortest paths starting with u or ending with v [57].

Betweenness of vertices has also been looked at in the PPI network with mixed results. Joy et al. examined the betweenness values of vertices in the PPI network, and found a significant number of vertices with high betweenness, but low degree, something not predicted by the standard scale-free model of PPI networks [58]. They suggest that these high betweenness are likely between complexes or other modules. Del Sol and O'Meara, however, looked at betweenness on vertices in protein complexes and found that many complexes had a vertex with high betweenness, suggesting that it was a central protein in the network [59]. Because these high betweenness vertices were found as part of complexes, this result cautions against simply removing high betweenness vertices in order to find complexes.

5.2.3.4 Clustering methods

Another common technique for finding modules in biological networks is a technique known as hierarchical clustering. In hierarchical clustering, there is a value between every two vertices that measures their similarity. The two most similar vertices are merged into a cluster, and the cluster receives a similarity measure with every vertex (or cluster). This process of merging the most similar vertices or clusters and calculating similarity values of the new cluster with existing elements is repeated until all elements are merged into a single cluster. The end result is a tree that can be divided to find the modules.

Hierarchical clustering was first used on genomic data in connection with gene expression and genetic interaction networks. In 1998, Eisen et al. gave some clustering methods that might be used for gene expression data and demonstrated its usefulness. Adarichev et al. used clustering methods

on the gene expression data in arthritis [60]. Tong et al. developed a large genetic interaction network [61], which has been used in subsequent interaction studies. They then clustered the network by using the number of shared neighbors as their measure of closeness. Parsons et al. used a similar method to Tong in order to cluster both genetic interaction data and gene-chemical interactions in order to find pathways that might be used in drug development [62].

Hierarchical clustering has also been applied to PPI networks to search for complexes and other modules. Rives and Galitski find modules in the PPI network by grouping vertices according to their shortest paths [63]. They give each edge a weight of 1, and as with the betweenness algorithms, calculate the shortest paths between all pairs of vertices. Unlike the betweenness algorithms, however, they are not concerned with the number of shortest paths a given vertex or edge might be part of, but with the relationships between vertices with short paths between them. For each pair of vertices, they calculate an association value, $\frac{1}{d^2}$, where d is the length of the shortest path between the vertices. They then do a hierarchical clustering algorithm: the algorithm starts by merging the two vertices with the highest association value. At each subsequent stage of the algorithm, the two entities with the highest association, be they vertices or clusters, are merged. The end result is a tree that can be divided to find the modules.

Like Rives and Galitski, Ravasz et al. also use hierarchical clustering to divide biological networks into modules [64]. Rather than using the shortest path to determine how closely two vertices are related, however, they use a property they call topological overlap, based on the number of shared neighbors that a vertex has. This is essentially the same as one of the mutual clustering coefficient metrics of Goldberg and Roth [30]. Otherwise, the algorithm is the same. It should be noted however, that the Ravasz algorithm is not designed specifically for protein complexes or even PPI networks, but for finding modules in all biological networks; the authors applied their algorithm to metabolic networks.

5.2.3.5 The core-attachment model

A small scale study by Dezso et al. [65] and a larger study by Gavin et al. [66] suggest that the proteins in a complex are not uniformly important but consist of a "core" of essential proteins as well as an additional "attachment" of less important proteins whose interactions with the complex may be transitory. Algorithms that use the core-attachment model first look for the core, then try to determine other proteins that may be attached.

One algorithm based on this model was developed by Leung et al. [67]. This algorithm starts by looking at pairs of vertices v_1 and v_2 with degrees d_1 and d_2. They look at whether or not there is an edge between v_1 and v_2 and how many neighbors they share. This is then compared to the probability of an edge between the vertices and the expected number of shared neighbors if the edges of v_1 and v_2 were randomly distributed, similar to the mutual clustering coefficient. This comparison gives a p-value $p(v_1, v_2)$ with lower p-values indicating a greater relationship than would be expected by chance.

The Leung algorithm finds cores by looking at p-values. Two vertices v_1 and v_2 are placed in the same core if $p(v_1, v_2)$ is lower than all other p-values involving v_1 or v_2. A vertex v is added to an existing core C if the largest p-value between v and a vertex of C is smaller than the smallest p-value between v and a vertex not in C. Once no more vertices can be added to the core, the attachments are found by looking for vertices with edges to at least half the core proteins.

A second algorithm based on the core-attachment model, COACH, was developed by Wu et al. [68]. COACH finds preliminary cores by looking at the neighborhood of a vertex v and designating as core vertices those vertices with high degree in the neighborhood. Preliminary cores are maximal subgraphs of core vertices with an edge density above a given threshold. The final list of cores is obtained by discarding preliminary cores that are too similar to cores that are already on the list. As in the Leung algorithm, the attachments are found by looking for vertices with edges to at least half the core proteins.

A major difference between the Leung algorithm and COACH is whether or not a protein

can be part of multiple cores. In the Leung algorithm, every protein is in at most one core. In the COACH algorithm, a protein with high degree has the potential to be in a core for each of its neighbors.

5.2.3.6 A minimum cut method

While we are unaware of any algorithms that use vertex-connectivity in the PPI network, there is one complex-finding method that includes minimum cuts and thus makes some use of edge-connectivity: Pržulj et al. applied the Hartuv and Shamir algorithm to connected components of the PPI network in order to find modules [69]. Hartuv and Shamir look for a minimum cut in a graph of n vertices. If the value of the cut is at least $\frac{n}{2}$, the graph is considered a module; otherwise, the algorithm is run recursively on each side of the cut until the condition is met. This leads to components that have an edge-connectivity at least half the number of vertices they contain. However, because the minimum edge-connectivity requirement for this algorithm is stated in terms of the number of vertices, this "edge-connectivity" measure is more closely related to the edge-density methods. It will produce small, dense modules; a subgraph could be k-connected for relatively high values of k, but if it was also large, it would not be found by this algorithm.

5.2.3.7 Other methods

There have been many other attempts to find complexes in the protein-protein interaction network using many different properties and models other than the ones discussed above. We give some well known methods here.

The Pereira-Leal et al. algorithm first converts the PPI interaction network into a line graph by switching the vertices and edges. An interaction from the original PPI network becomes a vertex in the line graph, with an edge between two vertices if they share a protein. The line graph gives the network more structure as well as allowing a protein to be part of multiple complexes. The line graph is then divided into modules using an algorithm based on flow simulation [70].

Chu et al. developed an algorithm to recover complexes from affinity purification data.

Their measure of closeness between two proteins was a statistic called the von Neuman diffusion kernel, which is based on the number of short paths between a pair of vertices. To calculate the diffusion kernel, the number of paths of a given length l is calculated, and this number is multiplied by a diffusion factor γ^l so that the importance of longer paths decays exponentially. The kernel is the limit of the sum of this number over all path lengths. This measure was then normalized and applied to a probabilistic model to determine the likelihood of two vertices being in the same complex.

Rungsarityotin et al. [71] also try to find complexes from affinity purification results. Rather than building an interaction, graph, however, they use Markov random fields directly on the interaction data. This allows them to more accurately model the possible errors in the interaction data as well as not relying on using either the hub-and-spoke or clique model for affinity purification.

Qi et al. used an artificial intelligence program to find complexes. They looked at known complexes and calculated several properties including edge density, various degree statistics, clustering coefficients, and the presence of various motifs including triangles and 4-cycles. These statistics were then compared with random collections of proteins and used to train a Bayseian Network [72].

5.3 My work

5.3.1 k-Connectivity and Independent Paths in the PPI Network

The concept of graph connectivity has not often been applied to protein-protein interaction networks. Hartuv and Shamir looked for subgraphs with n vertices that were $\frac{n}{2}$-connected, but because their threshold for connectivity was based on the number of vertices, their connectivity measure is largely a measure of edge density. There has not been an attempt to discover if connectivity might be a relevant statistic even if the graph in question has low edge density.

There are some reasons to be skeptical as to whether or not k-connectivity is meaningful in the context of PPI networks. The idea of k-connectivity is built around the notion of independent paths in a graph. Many of these paths may be very long. What is the meaning of a "path",

especially a long path, in the context of a PPI network? Does it have any meaning at all?

Despite these reservations, there are also some reasons to suspect that connectivity might be biologically meaningful in PPI networks. The first reason is that k-connectivity implies a certain amount of stability in a network. A network with high connectivity could remain connected even in the event of edge loss or vertex loss. Contrast this with edge density, which is simply a measure of the percent of pairs of proteins that interact; a graph could have an edge density of almost .5 while still being only 1-connected and therefore vulnerable to becoming disconnected in the case of a lost interaction. In subgraphs of PPI networks, stability may be important because it represents how durable the subgraph is in case of mutations. Mutations can cause either edge loss or vertex loss. Edge loss implies one of the proteins involved in an interaction mutates in such a way that the interaction can no longer occur, while vertex loss implies that a protein is either missing or so badly mutated that it cannot function.

We initially investigated connectivity in connection with protein complexes, where we felt that the stability of the overall complex would be an important feature. While it might at first appear that a mutation in one of its component proteins might cause a complex to become non-functional, the complex still might be able to function if it could maintain its basic structure. In the case of a mutation that prevents an interaction from occurring (edge loss), two adjacent proteins may no longer be able to bind, but the basic form of the complex may still remain the same due to other connections. Even mutations that cause the loss of a protein may be survivable as long as the basic structure of the complex remains intact. Some experimental studies have been done on complexes missing one or more proteins [73, 74, 75, 76]. While in some of these studies, the partial complex was not able to function at all, in other cases the partial complexes demonstrated at least partial function. In some cases, submodules of complexes were able to form independent of the rest of the complex [77] and even perform some functions [74].

The stability of a subgraph might also be important in signaling networks. In signaling networks, the loss of a connection or protein might cause one signal path to fail, but if there are multiple independent paths, the signal could still be passed through the network. Stability might

also be important in other molecular machines or biological processes. In general, stability would be a factor in any subgraph of the network where the fact that the subgraph is connected is important and it is possible that the subgraph might be able to function without all of its edges or vertices.

The second reason that connectivity and the presence of independent paths might be significant is that even if the paths themselves are not meaningful, they may be indicative of things that are. For example, signaling pathways often include many redundant pathways between the different layers of the network. These paths are short, but not independent. When looking at these paths in the undirected protein interaction network, they will appear as a large number of long independent paths between all vertices in the network. Despite the fact that these paths themselves are not used in the network, their presence is an indication of significance.

A similar argument can be made for complexes. If each protein in the complex binds to some number of adjacent proteins, then as the number of proteins in the complex increases, the edge density will decrease because it isn't possible for a single protein to bind to all others. The k-connectivity, however, will stay roughly constant as long as each protein remains bound to roughly the same number of neighbors. Thus, the k-connectivity would be significant even if the individual paths were not.

Finally, research on articulation points of a PPI network has suggested a connection between essential proteins and connectivity. Articulation points are vertices whose removal disconnects the network. There has been some experimental evidence suggesting that articulation points of any component in the PPI network are more likely to be essential proteins; mutations in these proteins tend to prove lethal to the organism [69]. This suggests that connectivity and the presence of paths does play a role in the network, even if we don't fully understand what that role might be.

In general, PPI networks are not connected. They do, however, have a large connected component, and subgraphs of that component have higher connectivities still. In order to test the theory that connectivity would be indicative of biological significance, we wanted to find the **most highly connected subgraph (MHCS)** of the network. While the literature contains some discussion of k-connected subgraphs for fixed values of k, I did not find an algorithm to find the

MHCS of a graph.

5.3.2 Looking for an MHCS in PPI networks

We developed an algorithm to find a most highly connected subgraph (MHCS) and applied it to the yeast and human PPI networks. We found several different subgraphs with high connectivity values that appear to have biological significance. The subgraphs that we found were distinct from those found by previous methods to find complexes or other functional groups based on edge density and similar statistics. Previous algorithms found modules that were small and dense. We found some of these small and dense modules as well, because large cliques have both high edge density and high k-connectivity. However, we also found sparser modules that were much larger than those found by other algorithms. The most highly connected subgraph of the yeast Y2H graph was a 16-connected subgraph, consisting of 49 membrane proteins. This subgraph has not been identified by any previous method.

5.3.3 Protein complexes

While the common theory says that protein complexes should appear as cliques or "near-cliques" in the PPI network, there are reasons to doubt this theory especially in the larger complexes. There is no reason why two proteins on opposite sides of a large complex should necessarily interact directly. We hypothesized that graph connectivity might be a better indicator of complexes, because it is important for a complex to retain its structure even if a protein has a mutation that causes it to lose an interaction (edge connectivity) or is missing altogether (vertex connectivity). Also, each protein in a complex is likely to interact directly with several nearest neighbors, which would result in high k-connectivity.

In addition, most of the previous algorithms for finding protein complexes use data about known protein complexes to evaluate their performance. However, the actual complexes do not influence the parameters used in the algorithm. In other words, these algorithms are based on intuition of what protein complex topology ought to look like or tuned to best find complexes

rather than being based on a systematic examination of the properties of actual complexes. The only instance of which we are aware where known protein complexes were used to determine which parameters should be used to find new complexes was in Qi et al. [72], summarized in Section 5.2.3. Before developing further complex-finding algorithms, we believed that we should have a better understanding of what to look for and which properties differentiate complexes from other subgraphs of the PPI network.

In order to assess the common theory as well as test the hypothesis that connectivity is a useful property in looking at complexes, we conducted a principled survey of the known protein complexes in *Saccharomyces cerevisiae* and looked at several different graph theoretic properties including edge density and k-connectivity. Our survey suggests that the edge density of complexes is not as high as the clique theory would imply and that other properties may be better indicators of protein complexes. Graph connectivity is one of those properties. While most complexes were not even 2-connected, this was generally due to a small number of degree 1 vertices (these may be the "attachments" from the core-attachment model). We used the most highly connected subgraph algorithm to find the MHCS of each complex and found that many complexes contained a 3-connected or 4-connected subgraph. In a survey of random comparable subgraphs of the PPI network that we called **pseudocomplexes**, very few pseudocomplexes had a 3-connected subgraph, and almost none had a 4-connected subgraph. We therefore believe that vertex connectivity will be a valuable tool in future development of complex-finding algorithms.

Chapter 6

Edge and Vertex Connectivity in the Protein-Protein Interaction Network

As mentioned in Chapter 5, neither vertex nor edge connectivity has been used much in the study of PPI networks. It is far more common to determine clusters in the interaction data based on edge density. For reasons mentioned in Chapter 5, however, we theorize that in some biological processes, connectivity may be a better indicator of a biological module than edge density. There is some evidence to support this in certain modules. For example, the interactions between proteins in the 20S proteasome [78] form the 4-connected graph shown in Fig.6.1 whose edge density is only 0.35. This edge density is far too low to be considered significant. The fact that it is 4-connected, however, is significant; it is extremely unlikely that a randomly chosen set of proteins would be 4-connected. It is far more likely that this subgraph would be correctly identified as a module by an algorithm searching for connectivity.

The study of connectivity in the PPI network is complicated by the fact that the network as a whole is not connected. It does, however, have a large connected component, and there are many subgraphs with even higher connectivity values. We find these subgraphs by iteratively finding a most highly connected subgraph (MHCS) of a given network.

A subgraph H of a graph G is a most highly connected subgraph if H is k-connected for some integer k and G does not have a subgraph that is $(k+1)$-connected. It should be noted that an MHCS H is not necessarily a maximum clique of a graph G, or even a clique at all. In fact, H can be quite sparse but still have a high connectivity value. While finding a maximum clique of a graph is an NP complete problem, a $MHCS$ can be found in polynomial time.

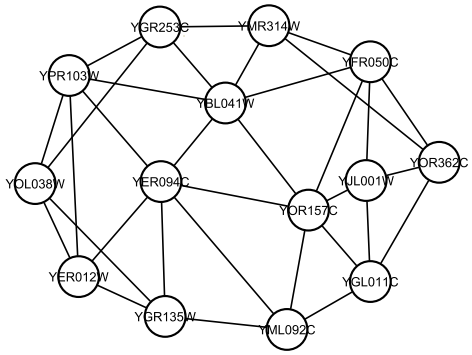


Figure 6.1: A graph representing the interactions between the proteins in the 20S proteasome. The graph has an edge density of only 0.35, but is 4-connected.

6.1 The Most Highly Connected Subgraph Problem

A search of the literature found some previous discussion of subgraphs with connectivities distinct from the connectivity of the parent graph. Multiple papers have discussed looking for k-connected subgraphs for fixed values of k [5, 79, 80, 81]. Matula [80] examined subgraphs of various connectivities. He referred to the connectivity of the most highly connected subgraph of a graph G as the **strength** of the graph and notated it as $\sigma(G)$. In the same paper, Matula looked at subgraphs that he called **clusters**, k-edge-connected subgraphs that do not contain $(k+1)$-edge-connected subgraphs, including clusters with connectivity $\sigma(G)$. Matula also discussed what he called **slicings** of a graph, partitions of the edges into cuts. He showed a relationship between k-connected subgraphs, clusters, the strength of the graph, and certain types of slicings that he called **narrow slicings**. He gave an algorithm to find a narrow slicing of a graph, along with some discussion of how this algorithm might be used to find k-connected subgraphs, clusters, and the strength of the graph.

There has also been discussion of the idea of "graph communities," where the community of a subgraph H in a graph G is the most highly connected subgraph of G that contains H [82].

Algorithmically, the work most similar to our problem is the Hartuv and Shamir edge connectivity clustering algorithm mentioned in the previous chapter. They cluster biological networks by looking for what they call **highly connected subgraphs**, subgraphs with connectivity greater than $n/2$, using an algorithm very similar to ours [83]. However, as mentioned in Chapter 5, because their definition of "highly connected" relies on the number of vertices, it is much closer to an edge density measure than to our metric; it is possible that the components would actually have a lower connectivity than the graph as a whole. Also, their algorithm tends to find many small subgraphs while our algorithm often finds much larger subgraphs.

We developed algorithms to find both a most highly edge-connected subgraph and a most highly vertex-connected subgraph. In cases where there were multiple most highly connected subgraphs, our algorithm returned an MHCS with the maximum possible number of vertices. We ran

these algorithms on large PPI networks and on graphs representing protein complexes. In both cases, our results are interesting and biologically relevant. Our results on complexes are discussed in the next chapter. In this chapter, we give the algorithm for finding an MHCS and discuss the MHCS of the overall yeast Y2H graph.

6.2 The Algorithm

We first give the algorithm for finding a most highly edge-connected subgraph in a simple graph. We then modify the algorithm to work on multigraphs or to search for a most highly vertex-connected subgraph.

The overall algorithm (Algorithm 1) takes a graph G and returns two values, a subgraph of G and the connectivity of that subgraph. Algorithm 1 will return a subgraph with the highest connectivity value; if two or more subgraphs have the same connectivity, the one returned will be one with the greatest number of vertices. The basis for the algorithm is a procedure described in Algorithm 2 that works by taking both a graph G and an integer k. Algorithm 2 looks for an MHCS of G assuming that G has a subgraph with connectivity at least k; we are uninterested in subgraphs that aren't at least k-connected. In order to search for these subgraphs, we find a minimum cut of the graph, then recursively run the algorithm on the subgraphs on each side of the cut. Algorithm 1 calls Algorithm 2 on the entire graph with a k value of 0.

Algorithm 1 $MostHighlyConnectedSubgraph(G)$

```
return MaxConSubgraph(G, 0)
```

Algorithm 2 $MaxConSubgraph(G, k)$

```
if |V| ≤ k then
  Let G_0 be an empty graph
  return (G_0, 0)
end if
k', H_1, H_2 ← MINCUT(G)
k_next ← MAX(k, k' + 1)
G_1, k_1 ← MaxConSubgraph(H_1, k_next)
G_2, k_2 ← MaxConSubgraph(H_2, k_next))
return MAX((G, k'), (G_1, k_1), (G_2, k_2))
```

We use a minimum cut algorithm that gives both the value of the cut, k', and the subgraphs on both sides of the cut, H_1 and H_2. We next calculate the minimum connectivity that we would be looking for in a subgraph, k_{next}, as the higher of k or $k' + 1$. Note that it is possible that a subgraph could have a smaller minimum cut than its parent graph, so this calculation is necessary to ensure that the value of k never decreases. We then recursively call the algorithm on H_1 and H_2 with k values in both cases of k_{next} in order to look for a subgraph of higher connectivity.

Finally, we return the maximum of (G, k'), the value returned by the recursive call on H_1, and the value returned by the recursive call on H_2. Here, we define "maximum" by ordering the subgraphs first by connectivity, then by the number of vertices.

There is one trivial case of the algorithm worth mention here: the case where G consists of a single vertex. Single vertices are sometimes considered "connected," such as when looking at the connected components of a graph. However, because the definitions of k-edge- and k-vertex connectivity require at least 2 vertices, for our purposes we do not consider a single vertex to be 1-connected.

Readers familiar with the Hartuv and Shamir HCS Algorithm [83] will note a strong similarity between that algorithm and Algorithm 2. Both algorithms search for a minimum cut in a graph or component of a graph, remove the edges that cross the cut, then recursively call the algorithm on each side of the cut. However, the differences between the algorithms, though small, are significant. The two algorithms have different goals. The HCS Algorithm partitions the graph into many small subgraphs. Our algorithm finds one particular subgraph. The HCS Algorithm searches for subgraphs with connectivity greater than $\frac{n}{2}$, then returns those subgraphs when they are found. Our algorithm requires us to continue searching until we are certain that there is no more highly connected subgraph. Our algorithm also contains an additional step after the recursion where we compare subgraphs in order to find the most highly connected, while the HCS Algorithm contains no further steps after the recursion. Our algorithm also requires a proof of correctness to insure that it does in fact return a MHCS. Because the only requirement of the HCS algorithm is that the subgraphs it finds are $\frac{n}{2}$-connected, it is simple enough that the algorithm itself suffices without

any further proof of correctness. The results given by the two algorithms are different, with the HCS finding smaller subgraphs while our algorithm is more likely to find large subgraphs. The subgraphs found by our algorithm are also likely to have higher connectivity values; as mentioned earlier, because the connectivity that the HCS algorithm looks for is based on the number of vertices in the subgraph, its idea of "highly connected" is actually more closely related to edge density.

There are also strong similarities between our algorithm and the Matula Narrow Slicing Algorithm [80]. The narrow slicing algorithm partitions the edges of a graph G into a series of cuts $C = \{C_1, \ldots, C_m\}$ such that each C_i is a minimum cut of a connected component of $G - \bigcup_{j=1}^{i-1} C_j$. Like both Algorithm 2 and the HCS Algorithm, the Narrow Slicing Algorithm repeatedly looks for minimum cuts, then divides the graph by removing edges that cross the cut. The Narrow Slicing Algorithm, however does this iteratively on the connected components of the graph rather than recursively. The stopping conditions of the algorithm are different. The Narrow Slicing Algorithm is designed to partition the edges into a series of minimum cuts; it continues creating cuts until every edge is included in some cut, then returns the entire set of cuts C with no further modifications or comparisons. Matula proved that the strength of G is equal to the cardinality of the largest cut in C, and that each cluster is a connected component divided by some cut in C. It would therefore be possible to find an MHCS using the Narrow Slicing Algorithm, but this is not the primary goal of the algorithm and would require additional steps.

6.2.1 Proof of Correctness

By definition, the empty graph is 0-connected, so every graph must have at least a 0-connected subgraph. Therefore, the entire algorithm will work correctly if $MaxConSubgraph(G, 0)$ correctly returns an MHCS with connectivity greater than or equal to 0.

We use induction to prove that if G has a subgraph that is at least k-connected, $MaxConSubgraph(G, k)$ will return an MHCS of G with the greatest number of vertices; if G does not have a subgraph that is at least k-connected, $MaxConSubgraph(G, k)$ will return some subgraph of G with a correct connectivity value, but it may not be an MHCS.

For the base case, for any value of k, consider a graph with at most $k+1$ vertices. There are two possible cases here. If the graph is a complete graph on $k+1$ vertices, then it is k-connected and does not have a more highly connected subgraph. A minimum cut with value k will be found, which by Menger's theorem is also its edge-connectivity ([2]; summarized in [84]). The recursive step will then call the algorithm on $(H_1, k+1)$ and $(H_2, k+1)$ where both H_1 and H_2 are proper, non-empty subgraphs of G. Because both H_1 and H_2 will have fewer than $k+1$ vertices, the recursive steps will return empty subgraphs with connectivities of 0, causing the entire algorithm to correctly return that G is its own MHCS with connectivity k. If the graph is not a complete graph on $k+1$ vertices, it is not k-connected and has no k-connected subgraph. In this case, the MINCUT subroutine will return its minimum cut k' (which again by Menger's theorem is its edge-connectivity). Because G is not k connected, $k' < k$, so $k_{next} = k$, and the recursive step will call the algorithm on (H_1, k) and (H_2, k). As in the previous case, because both H_1 and H_2 have at most k vertices, the recursive steps will return empty subgraphs with connectivities 0, causing the entire algorithm to return G with its connectivity of $k' < k$.

For the inductive step, assume that for any value of k and any graph G with $k+i$ or fewer vertices, the algorithm will perform correctly. Look at a graph G with $k+i+1$ vertices. We find its minimum cut (H_1, H_2), which gives us the connectivity of G, k'. If G has a subgraph H with connectivity greater than the connectivity of G, then H must lie entirely on one side of the cut; otherwise, we can take the cut ($H \cap H_1$, $H \cap H_2$) in H, which will be less than or equal to (H_1, H_2), contradicting that H has a greater connectivity than G. Therefore, one of the following must be an MHCS of G: G itself, an MHCS of H_1, or an MHCS of H_2. Since both H_1 and H_2 have fewer vertices than G, and $k_{next} \geq k$, $MaxConSubgraph(H_1, k_{next})$ and $MaxConSubgraph(H_2, k_{next})$ will both return a correct value by the inductive hypothesis. If G was k-connected, the algorithm will return either G itself or, in the case that one of the recursive calls found a more connected subgraph, the larger of the two; if there was a tie, our definition of "maximum" will cause it to return a subgraph with the highest number of vertices. If G was not k-connected, then the algorithm will return either G and its connectivity, or the largest of $MaxConSubgraph(H_1, k)$ and

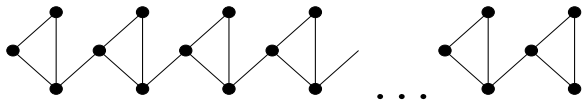


Figure 6.2: A worst case example. It's possible that at each stage of the algorithm, the minimum cut found is one that divides the graph into subgraphs of 3 and $n-3$ vertices. In this case, it would take $n/3$ recursive calls to find a most highly connected subgraph.

$MaxConSubgraph(H_2, k)$, both of which will either return an MHCS of connectivity at least k, or a subgraph with a correct connectivity. In either case, the algorithm returns correctly.

6.2.2 Complexity

The algorithm depends on the efficiency of the algorithm used to obtain the minimum cut. Let k_{min} be the weight of a minimum cut and k_{max} be the connectivity of the MHCS. Gabow has an algorithm that calculates the min cut in $O(m + k_{min}^2 n log \frac{n}{k_{min}})$ [85]. In the worst case, the recursive step might need to be run $O(n)$ times. See the example in Fig.6.2. This gives us a total running time for the algorithm of $O(nm + k_{max}^2 n^2 log \frac{n}{k_{max}}))$ if we use Gabow's algorithm. Note that this time bound is not tight and could possibly be improved by further analysis.

6.2.3 Haircut

We have made an improvement to the basic algorithm that we call the haircut step, similar to the haircut operation of MCODE [49] except instead of removing vertices of degree 1 we allow vertices with degrees less than an arbitrary k to be removed. In $MaxConSubgraph(G, k)$, before we call the minimum cut function, we recursively remove all vertices of degree less than k, leaving behind a k-core, a graph where all vertices have degree at least k. We call this operation, shown in Algorithm 3, the haircut and refer to the subgraph it produces as G's k-haircut. The Hartuv and Shamir HCS Algorithm [83] also includes the idea of removing low-degree vertices. However, because the HCS algorithm does not know in advance the connectivity for which it is looking, the algorithm may inadvertently remove vertices that are part of a highly connected subgraph. In our algorithm, by contrast, because we have a minimum connectivity k for which we are searching, we

can safely remove vertices with degree less than k without affecting the correctness of the algorithm.

Although the haircut is not guaranteed to improve the time bound of the algorithm, we have found in practice that it considerably speeds up the performance by shrinking the number of vertices that we need to work with and also lessening the possibility that we will find a trivial minimum cut later in the algorithm. Also, if the haircut eliminates all vertices in the graph, we can end our algorithm there without needing to look further for a minimum cut.

Algorithm 3 $Haircut(G, k)$

```
toCheck ← V
while toCheck ≠ ∅ do
  A ← ∅
  for all v ∈ toCheck do
    if v ∈ V and Deg(v) < k then
      for nbr ∈ Adj(v) do
        Add nbr to A
      end for
      Delete v from G
    end if
  end for
  toCheck ← A
end while
```

Note that in line 5 of the algorithm, it is necessary to check that $v \in V$. It is possible that v may have been added to the list *toCheck* in an earlier iteration of the loop, but deleted from V before it reached this check. Thus, in order to avoid an error, we verify that $v \in V$ before proceeding.

The haircut is done as the first step of the $MaxConSubgraph$ algorithm, leaving us with the modified version of Algorithm 2 shown in Algorithm 4.

6.2.3.1 Proof of Correctness

Only a few modifications need to be made to the previous proof. In the base case, if we have a complete graph with $k+1$ vertices, the haircut will not remove anything, we will find a minimum cut of value k, then on the subsequent recursive call, both graphs will have degree less than k, and all vertices will be eliminated by the haircut, causing the recursive step to return empty graphs of

Algorithm 4 $MaxConSubgraph(G, k)$

```
Haircut(G, k)
if V = ∅ then
  return (G, 0)
end if
k', H1, H2 ← MINCUT(G)
k_next ← MAX(k, k' + 1)
G1, k1 ← MaxConSubgraph(H1, k_next)
G2, k2 ← MaxConSubgraph(H2, k_next))
return MAX((G, k'), (G1, k1), (G2, k2))
```

connectivity 0 and the entire algorithm to return the complete graph with connectivity k. If the graph is not complete or has fewer than $k+1$ vertices, all vertices will be eliminated by the haircut, causing the algorithm to return an empty graph with connectivity 0.

In the inductive step, simply note that any vertex with degree less than k cannot be part of any subgraph that is at least k-connected. Therefore, the removal of these vertices does not affect the algorithm in the case that there is a k-connected subgraph. All other steps of the proof are the same.

6.2.3.2 Complexity

The worst-case example in Fig.6.2 is not helped by the haircut, so we know that the haircut cannot decrease the worst-case analysis of the timing of the algorithm. We also show that it does not increase it.

The haircut operation to remove all vertices of degree k or less takes $O(kn)$ time. In the initial pass through the algorithm, all vertices must be examined to check if they have a degree less than k. In subsequent passes, however, only those vertices adjacent to vertices deleted in the previous pass must be examined; if the previous step deleted n_i vertices, the algorithm will have to check at most $(k-1)n_i$ vertices. Because each vertex can only be deleted once, this will lead to a maximum of $n + (k-1)n = kn < O(nm)$ steps. Because this is less than the time taken for the minimum cut phase, adding the haircut does not affect the overall complexity of the algorithm.

6.2.4 Variants on the Basic Algorithm

We consider two variants on the basic algorithm: edge-connectivity on multigraphs and vertex-connectivity.

Very few changes are required in the basic algorithm in order for it to work on multigraphs. Both the haircut and the Gabow minimum cut algorithm will work on multigraphs. If using the variant of the algorithm with the haircut, no changes to the algorithm are required. If using the variant of the algorithm without the haircut, the base case of the recursion needs to be changed to $|E| < \frac{|V|k}{2}$ rather than $|V| \leq k$ because a multigraph with k or fewer vertices can still be k-connected. In both cases, the proof of correctness needs to be modified so that the base case is a graph of k edges or fewer.

Vertex connectivity presents a greater challenge. There are two issues that must be considered. The first is that a cut does not cleanly divide the graph into two pieces; in addition to the vertex sets on both sides of the cut, we also have the vertices that are part of the cut. We will want these vertices to be included in both subgraphs. The second issue is that vertex cut algorithms do not work well on complete graphs because they require both sides of the cut to be non-empty. For our purposes, we will define the vertex connectivity of a complete graph with n vertices to be $n-1$, but we will need to deal with complete graphs separately from other cases. Dealing with these issues requires further modifying Algorithm 2 to the procedure shown in Algorithm 5.

Algorithm 5 $MaxVConSubgraph(G, k)$

```
Haircut(G, k)
if V = ∅ then
  return (G, 0)
end if
if G is a complete graph then
  return (G, |V| − 1)
end if
k', H1, H2, C ← MINVCUT(G)
k_next ← MAX(k, k' + 1)
G1, k1 ← MaxVConSubgraph(G[V_H1 ∪ C], k_next)
G2, k2 ← MaxVConSubgraph(G[V_H2 ∪ C], k_next))
return MAX((G, k'), (G1, k1), (G2, k2))
```

As before, k' represents the value of the minimum cut, and H_1 and H_2 represent the subgraphs on either side of the cut, but we also have a fourth return value C which is those vertices that are part of the cut. Those are added to the vertex sets of H_1 and H_2. We make the recursive calls on the subgraphs induced by the two vertex sets $V_{H_1} \cup C$ and $V_{H_2} \cup C$, $G[V_{H_1} \cup C]$ and $G[V_{H_2} \cup C]$.

The proof of correctness is still much the same. In the base case, we look at a graph with $k+1$ or fewer vertices. If the graph is not a complete graph with $k+1$ vertices, it will still be emptied by the haircut, and if it is a complete graph with $k+1$ vertices, it will be returned with a connectivity k by our check for complete graphs. In the inductive step, we assume that the algorithm returns correctly if the graph has $k+i$ or fewer vertices and examine a graph with $k+i+1$ vertices. The proof of this step remains the same as in the edge connectivity case as long as we note that neither H_1 nor H_2 can be empty. Thus, the size of the graph decreases with each recursive step, while k either increases or remains the same. Therefore, the remaining steps of the proof are still valid. Both recursive calls to the algorithm will return correctly, and the comparison will give us an MHCS with connectivity at least k if such a subgraph exists.

When looking at the running time for $MaxVConSubgraph$, the haircut still takes kn steps, and checking for a complete graph takes m steps, so the running time still depends on the speed of the minimum cut algorithm. Gabow gives a vertex cut algorithm that runs in $O(k_{min}n(n + min\{k_{min}^{\frac{5}{2}}, k_{min}n^{\frac{3}{4}}\}))$ [86], where k_{min} is the size of the minimum vertex cut. The number of recursive calls needed is not as obvious as in the edge connectivity case because of the fact that vertices in the cut are present in both recursive calls. A similar example to the one in Fig.6.2 shows that $O(n)$ is a lower bound for the maximum number of recursive calls. We will prove that $O(n)$ is also an upper bound on the worst case. This bound on the number of recursive calls gives a time bound of $O(k_{max}n^2(n + min\{k_{max}^{\frac{5}{2}}, k_{max}n^{\frac{3}{4}}\}))$, where k_{max} is the connectivity of the MHCS.

Lemma 18. *Algorithm 5 requires a maximum of $O(n)$ recursive calls when run on a graph of n vertices.*

Proof. Look at the recursion tree formed by the recursive calls to the function, so the initial call

to the function is the root, the calls it makes are its children, the calls its children make are its grandchildren, etc., and the base cases are the leaves. This tree is a binary tree, and so if it has ℓ leaves, then there are $2\ell - 1$ nodes in the graph. We will show that the number of leaves, and thus the total number of recursive calls, is $O(n)$. To show this, we make the following claim.

Claim 2. *Let $N(G, k)$ be the node representing a call to the function on a graph G searching for a subgraph of connectivity at least k.*

a) If $|G| \leq k$, $N(G, k)$ is a leaf.

b) If $|G| = k + r$ for some $r > 0$, $N(G, k)$ is the ancestor of at most $r + 1$ leaves.

To show the first part of the claim, simply note that if G has k or fewer vertices, all vertices will be eliminated by the haircut, and the algorithm will return without making any further recursive calls. Thus, the node representing such a call is a leaf.

The second part of the claim will be proved inductively. For the base case, we look at $r = 1$. For a graph with $k + 1$ vertices, if the graph is not complete, the haircut will eliminate all vertices, and the call will be a leaf. Otherwise, the algorithm will find a minimum cut, then make two recursive calls on graphs G_1 and G_2, each of which will have fewer vertices than G. Because $k_{next} \geq k$, $|G_1| \leq k_{next}$ and $|G_2| \leq k_{next}$. By the first part of the claim, then, $N(G_1, k_{next})$ and $N(G_2, k_{next})$ are both leaves, and $N(G, k)$ is the ancestor of 2 leaves.

For the inductive step, assume that for all $j \leq i - 1$, a node representing a call on a graph with $k + j$ vertices is the ancestor of at most $j + 1$ leaves. Look at $N(G, k)$ with $|G| = k + i$. Assuming that the call is not itself a leaf, the algorithm will find a minimum cut and divide the graph into two subgraphs, G_1 and G_2. The algorithm will then make recursive calls on (G_1, k_{next}) and (G_2, k_{next}) where $k_{next} \geq k$. Suppose that one of the subgraphs, WLOG say G_1, has fewer than $k_{next} + 1$ vertices. Then, $N(G_1, k_{next})$ is a leaf, and because $|G_2| < |G|$ and $k_{next} \geq k$, the inductive hypothesis implies that $N(G_2, k_{next})$ is the ancestor of at most i leaves. Thus $N(G, k)$ is the ancestor of at most $i + 1$ leaves. Now suppose that both $|G_1|$ and $|G_2|$ are greater than k_{next}. Let C be a minimum vertex cut of G with H_1 and H_2 the subgraphs on either side C. Let $|C| = c$,

$|H_1| = h_1$, and $|H_2| = h_2$. There are two cases depending on the size of C.

Case 1: $c = k - x$ vertices for some $x > 0$.

In this case, we have $i + x$ vertices that are not part of the cut. Thus, $|G_1| = k - x + h_1$ and $|G_2| = k - x + h_2$ for some $h_1 + h_2 = i + x$ and $h_1, h_2 > 0$, and $k_{next} = k$. By the inductive hypothesis, $N(G_1, k_{next})$ is the ancestor of at most $h_1 - x + 1$ leaves and $N(G_2, k_{next})$ is the ancestor of at most $h_2 - x + 1$ leaves. This implies $N(G, k)$ is the ancestor of at most $h_1 + h_2 - 2x + 2 = i - x + 2$ leaves, which given our initial assumption $x > 0$, is less than or equal to $i + 1$.

Case 2: $c = k + x$ vertices for some $x \geq 0$.

In this case, we have found a cut of size at least k, so the connectivity of the subgraph we are looking for in the next iteration will change to $k_{next} = k + x + 1$. We will have $i - x$ vertices not in the cut. Therefore, the two subgraphs will have $|G_1| = k_{next} + h_1 - 1$ and $|G_2| = k_{next} + h_2 - 1$ for some $h_1 + h_2 = i - x$ and $h_1, h_2 > 0$. By the inductive hypothesis, $N(G_1, k_{next})$ is the ancestor of at most c_1 leaves and $N(G_2, k_{next})$ is the ancestor of at most c_2 leaves. This implies $N(G, k)$ is the ancestor of at most $c_1 + c_2 = i - x$ leaves, which is less than $i + 1$ given our assumption $x \geq 0$.

This completes the proof of the claim. Thus, if our root is a call on a graph with n vertices and $k = 0$, we can have at most n leaves, which implies fewer than $2n$ calls to the function, and the bound is proved.

□

6.3 Highly Connected Subgraphs in Yeast Two-Hybrid Networks

We obtained protein-protein interaction data from Biogrid [87] and ran our algorithms for both edge and vertex connectivity on the yeast Y2H interaction graph and the human Y2H interaction graph. Both algorithms produced exactly the same result, so we give only the results for the stronger condition, vertex connectivity.

We implemented algorithms to find both the most highly edge-connected and most highly vertex-connected subgraphs. In both cases, we implemented the variants of the algorithms that use the haircut. We implemented the algorithms to work on simple graphs, though in both cases

they could be modified to work on multigraphs. In implementing the algorithms, we used the edge connectivity algorithm of Stoer and Wagner, which is a simplification of the Nagamochi and Ibaraki algorithm [88]. The vertex connectivity algorithm we used is the Even and Tarjan connectivity algorithm [89]. Our implementation is based upon transforming the graph G into a directed graph G' where each vertex v from the original graph becomes two vertices in the new graph, v_{in} and v_{out}, with an edge of capacity 1 between them. The implementation then calculates the minimum cut of G from one of its vertices $v \in V$. This will give the global minimum cut unless v happens to be a part of all minimum cuts. In order to ensure that we find a global minimum cut, therefore, we must run the algorithm multiple times, until the number of times we have run the algorithm is greater than the calculated value for the minimum cut, k_{min}.

6.3.1 Testing the Implementation

6.3.1.1 Small-Scale Testing

We first tested the algorithms on the small graphs shown in Fig. 6.3. For the initial testing of the algorithms, these small graphs had several advantages over larger ones. These small graphs have an obvious most highly connected subgraph. The algorithms runs quickly on these graphs, so we can have multiple runs of the algorithms in order to determine any errors in the implementation. The small size of the graphs allows us to print debugging information which can be interpreted by a human operator. Finally, because of the small size of the graphs, we were able to check the input files by hand to be sure that any errors we encountered were errors in the implementation rather than errors in the input.

Each of the graphs in Fig. 6.3 was designed to test a different aspect of the algorithm. The graph in Fig. 6.3(a) was designed to test the haircut function's ability to remove vertices of various degrees, as well as to insure that the recursive aspect of the haircut worked correctly. The pentagon in Fig. 6.3(b) insures that the algorithm will work correctly even if a graph is its own MHCS. The graph in Fig. 6.3(c) is designed to test how the algorithm handles multiple subgraphs with the

same connectivity and insure it returns the one with the most vertices. The bow-tie graph in Fig 6.3(d) serves the same function as (b) in testing the edge-connectivity version of the algorithm. However, the bow-tie is designed primarily to test the vertex connectivity version of the algorithm and insure that it finds the most highly vertex-connected subgraph even if that is different from the most highly edge-connected subgraph. The graph in Fig. 6.3(e) is also designed primarily for the vertex connectivity version of the algorithm, insuring that the algorithm can handle a vertex cut that divides the graph into more than two connected pieces. The graph in Fig. 6.3(f) was designed to see if the algorithm would work when the order of the vertices in the input file was permuted, something that unfortunately does not show up in the figure; the input file for this graph listed the vertices in a different order from the order they appeared in the edge list. Finally, the graph of Fig. 6.3(g) is designed to be a general test of the implementation, insuring it can find highly connected subgraphs with connectivities up to 4.

6.3.1.2 20-Connected Subgraphs

After testing the implementation on small graphs, we then tested its ability to find highly connected subgraphs in large graphs. We generated two 20-connected subgraphs. The first was a 21-clique. The second was a 95-vertex graph consisting of five 19-cliques linked together with 19 cycles. We then ran both the edge and vertex connectivity algorithms on these graphs in order to verify that the algorithms would return that these graphs were their own MHCS, and that they were 20-connected.

Next we embedded these subgraphs into a larger graph to see if the algorithm would find them. We started with the yeast Y2H network. This graph does not have a 20-connected subgraph, which was demonstrated by doing a 20-haircut and observing that it removes all vertices from the graph. We then added the vertices and edges of the 21-clique to the yeast Y2H graph, as well as adding 19 randomly chosen edges between the 21-clique and the Y2H graph. We ran both the vertex and edge connectivity versions of the algorithm on this new graph. Both algorithms successfully returned the 21-clique as the most highly connected subgraph. The process was repeated with the

95-vertex graph. Again, both algorithms were successful at finding the embedded 20-connected graph.

Finally, in order to be sure that the embedding procedure did not produce any unintended effects, we chose 21 proteins in the yeast Y2H network at random and added edges to make a 21-clique from these proteins. Again, both vertex and edge connectivity algorithms successfully returned this 21-clique as the most highly connected subgraph. We did the same thing with the 95-vertex graph, choosing 95 proteins at random and adding edges to make these proteins into a 20-connected subgraph. Once more, both vertex and edge connectivity algorithms successfully returned this 20-connected graph as the most highly connected subgraph.

6.3.2 Subgraphs with High Connectivity

We found several interesting subgraphs in the entirety of the yeast Y2H network. The most highly connected subgraph was a set of 49 membrane proteins that were 16-connected (Fig. 6.4). All interactions in this subgraph came from a study of membrane proteins [90]. This subgraph is much more highly connected than the next most highly connected subgraph, which is 8-connected, so we considered the possibility of study bias; the membrane study was biased to maximize the chance of observing an interaction. However, this 16-connected graph contains only a small subset of proteins in the study; other proteins in the study had a higher average degree than other membrane proteins in the composite Y2H network, but not high enough to expect such a highly connected subgraph. The study contained 535 membrane proteins which had an average degree of 7.26 (2.30 excluding the 49 proteins in the MHCS). Excluding this study left 154 membrane proteins with an average degree of 1.55. This leads us to conclude this subgraph is not simply the result of study bias. We examined GO annotations for these 49 proteins using the FuncAssociate program [91]. FuncAssociate gives shared functions of these proteins as well as a p-value that gives the probability that these shared functions would be seen by chance, and an adjusted p-value that takes into account the fact that we are looking at multiple hypotheses simultaneously. Of the 49 proteins, 48 had GO annotations; the remaining protein had no annotated function. GO annotations indicate that almost all of

these membrane proteins are related to the endoplasmic reticulum (34 proteins, p-value = 3.9e-29, adjusted p-value < .001) and that many are involved in transport processes (27 proteins, p-value = 1.4e-08, adjusted p-value < .001). We do not believe this subgraph represents a single complex, but it appears to be biologically significant, and we theorize that it may be either several overlapping complexes or a type of signaling network. Collaborating biologists find this subgraph intriguing and are analyzing it to discover its function and significance.

While the MHCS algorithm is only designed to find a single, most highly connected subgraph in any given graph, it is possible to find more subgraphs by eliminating the vertices in the MHCS and re-running the algorithm on the remaining graph. This allows us to find multiple subgraphs with high connectivities, but it precludes the possibility of finding overlapping subgraphs.

Excluding the 49 membrane proteins, the next most highly connected subgraph was an 8-connected subgraph consisting of 9 proteins, PAT1 and the 8 proteins LSM1-8 shown in Fig. 6.5. These are all proteins involved in the spliceosome, and it appears that this subgraph is the combination of two highly overlapping complexes. PAT1 and LSM1-7 form a critical decapping complex which also serves to protect the 3' end of mRNA [92]. LSM2-8 together with PRP24 form the protein components of the U6 snRNP, a complex comprised of both proteins and RNA; in the U6 snRNP, the LSM proteins form a ring with PRP24 at a set distance from them [93], so it is not surprising that PRP24 was not part of our subgraph.

The third most highly connected subgraph was a 6-connected subgraph of 9 proteins shown in Fig. 6.6. Eight of these proteins are contained in the DASH complex, a 10-protein complex which plays a role in the chromosome separation of mitosis by stabilizing the spindle as part of the kinetochore [94, 95]. The ninth protein, SLI15, is not known to be part of the DASH complex, but has a closely related function, related to spindle and kinetochore attachment [96].

The fourth most highly connected subgraph was a 5-connected subgraph with 131 vertices (Fig. 6.7). We were unable to find a unified GO annotation for this subgraph. Despite the fact that we were unable to find a unified function for this subgraph, we still believe that it may be significant. While a 5-connected graph is not as notable as a 16-connected graph, it is still unusual,

and the fact that there are 5 paths between any pair of vertices is likely significant. Because this subgraph contains 131 vertices and has an edge density of only 0.06, it is unlikely that any other algorithm would be able to identify this as a module.

We also ran our algorithm on the human Y2H network and found that the MHCS was a 7-connected graph of 52 proteins (Fig. 6.8). Due to the limited number of well-annotated human proteins, however, we were not able to discover the biological significance of this graph. The second MHCS was a 6-connected graph of 73 proteins (Fig. 6.9), and we were also unable find a common function for this graph. The third MHCS, however, was a 6-connected graph with 15 proteins (Fig. 6.10), which had a common function in DNA replication initiation (11 proteins, p-value = 4.8-23, adjusted p-value $< .001$). The fourth MHCS was 4-connected and had 23 proteins (Fig. 6.11), most of which were involved in transcription regulation (20 proteins, p-value = 6.2e-19, adjusted p-value $< .001$). They appeared to be mostly involved in the regulation of various sex hormones (3 were part of estrogen receptor signaling pathways while 9 were involved with steroid receptors). The fifth MHCS, 4-connected with 8 proteins (Fig. 6.12), also had common functions in transcription regulation activity (7 proteins, p-value = 2.6e-7, adjusted p-value = .001) and regulation of RNA metabolism (p-value = 9.1e-7, adjusted p-value = .001). We did not have any data on human complexes, so we were not able to compare these subgraphs to existing complexes.

6.3.2.1 Verification of Subgraph Connectivity

For all subgraphs whose connectivity could not be easily verified with the naked eye (the first and fourth most highly connected subgraphs in yeast and all human subgraphs), we verified the edge-connectivity of the subgraphs using three different algorithms: the Hao-Orlin [97], the Karger-Stein [98], and the Nagamochi-Ibaraki [88]. We used implementations of these algorithms done by Chekuri et al. [99] and available at `http://www.columbia.edu/~cs2035/code.html`. All three algorithms verified the connectivities reported above.

6.3.3 MHCS and Other Methods of Finding Modules in the PPI Network

The MHCS technique is not intended as a replacement for MCODE or other methods of finding modules in a network. Rather, it is meant to complement existing methods. Existing methods often find small, dense modules. These small modules are only occasionally found by looking for the MHCS of a graph. Instead, the MHCS technique is capable of finding large, relatively sparse graphs that would not be identified as modules by other methods. Many of these large, sparse subgraphs are likely meaningful, as evidenced by the fact that many of them contain proteins with a common function.

6.3.3.1 Comparisons with MCODE

We ran MCODE on the entire yeast Y2H network. MCODE successfully found the 9-clique associated with the U6 snRNP and the 9-vertex, 6-connected subgraph associated with the DASH complex but did not find the 16-connected subgraph or any significantly overlapping clusters, nor the 5-connected subgraph. In the case of the two subgraphs that MCODE found, however, it found the exact same subgraphs as our method.

When we run MCODE on just the 16-connected subgraph from the yeast network, however, it found the clusters shown in Fig. 6.4. There are two reasons why we would see clusters in the 16-connected subgraph when MCODE was given just that, but not see clusters that overlap with the 16-connected subgraph when MCODE is run on the entire yeast Y2H network. Perhaps the most important reason is that the weights on the vertices (described in Section 5.2.3.1) were determined using only vertices in the 16-connected graph, causing the vertices in the subgraph to have higher weights than they would have otherwise. The fact that seeds could only be chosen from within the 16-connected subgraph also helped MCODE find clusters entirely within this subgraph.

Running MCODE on the entire human Y2H network produced even less overlap with the MHCS algorithm than in the yeast. Of the five subgraphs discussed in the results section, only one, the fifth MHCS consisting of 8 4-connected proteins, had any significant overlap with the MCODE

results; MCODE found a subgraph with 6 proteins, all 6 of which were in the fifth MHCS. The other two proteins that our algorithm found, however, had multiple shared functions with the other proteins in the modules, suggesting that they ought to be considered with the others.

On both networks, MCODE did not find any of our large subgraphs. The largest subgraph of ours that was found by MCODE had only 9 vertices, while most of our subgraphs were very large, with the largest ones having 131, 73, 52, and 49 proteins. This suggests that our algorithm will find large subgraphs that MCODE will not.

6.3.3.2 Comparisons with Other Methods

While we have not compared the results of the MHCS algorithm with the results from any complex-finding algorithms other than MCODE, we have reason to believe that other algorithms would be even less successful at finding these large subgraphs. The 16-connected subgraph in the yeast network has an edge density of only 0.49, too low to be considered significant by most algorithms looking for edge density (for comparison, the King et al. algorithm looks for subgraphs with an edge density of at least 0.7 [54]). For other subgraphs not found by MCODE, the situation is similar. The 5-connected yeast subgraph has an edge density of only 0.06, and the four most highly connected subgraphs for human have edge densities of 0.23, 0.13, 0.61, and 0.24. In addition, many have too low a ratio between their connectivity and the number of vertices (16 vs. 49, 5 vs. 131, 7 vs. 52, 6 vs. 73, 6 vs. 15, 4 vs. 23) to be found by the Hartuv and Shamir technique.

Most of the other algorithms not based on edge density would also have trouble finding these subgraphs. Because of the way these subgraphs were found, there are many paths between their vertices, which could cause problems for algorithms based on betweenness; vertices within the subgraph would be on many short paths between other vertices in the subgraph. The presence of these paths could cause the algorithm to split the high-connectivity subgraphs. For example, simply within the 49-vertex 16-connected subgraph, there are 10 vertices with betweenness values higher than 20 and one with a betweenness of 33.0, all of which would be possible candidates for splitting the network. Clustering algorithms that work on methods other than betweenness tend

to find small, dense modules. Likewise, the "cores" found by algorithms using the core-attachment model are small by design, because they are meant to be part of a complex, and complexes are rarely as large as the modules we have found here. Because the significance of our large highly-connected modules is spread throughout the module, not only are these modules unlikely to have a small subgraph that would qualify as a "core", but it is unlikely that there are nodes within our highly-connected subgraph that would have a large number of edges to any small subgraph (potential core) of the module, as the "attachment" phase of these algorithms require.

6.4 Conclusion

We have developed a new algorithm to look for biologically significant subgraphs in protein-protein interaction networks. Connectivity is a common graph theory concept but has seldom been used to study biological networks. We show that the most highly connected subgraph of the overall Y2H network, as well as some other subgraphs with high connectivity, are biologically significant. Of the four yeast subgraphs discussed here, one corresponded to a known complex, another to a pair of overlapping complexes, and one subgraph whose proteins share GO annotations and whose exact function is being determined by biologists. The fourth subgraph has not yet been demonstrated to have a common function, but its large size makes it significant and shows that our algorithm is capable of finding subgraphs that might be overlooked by other methods.

The algorithm performed even better on the human Y2H network. While we could not find a common function for the two most highly connected subgraphs, the algorithm produced 3 other subgraphs that clearly shared a function, 2 of which were not found by other methods. It should be emphasized here that we believe that the two most highly connected subgraphs have a biological significance, but the fact that only a small fraction of the functions of human proteins have yet been determined hinders us in our attempts to classify these large subgraphs of human proteins.

The algorithm does have some drawbacks, most notably the fact that our algorithm does not cluster the PPI network. The yeast PPI network does not contain multiple maximal subgraphs for a given connectivity. While our algorithm can be modified to give more than one subgraph by

removing the vertices of the MHCS and running the algorithm again, this method still gives only a small number of relevant subgraphs. Of the subgraphs we investigated in the PPI network, only the first three clearly appeared to be biologically significant; the fourth was too large to easily see any common themes, but the fact that our algorithm was able to find such a large subgraph and identify it as a module is noteworthy and suggests that our algorithm may be able to find large modules that would go undetected by other methods that are primarily based on edge density. We feel that our algorithm will be a valuable tool for studying PPI networks. The algorithm found the MHCS of the Y2H network, which is a biologically significant subgraph that wasn't found using existing methods. The algorithm can also be run on subsets of proteins, such as co-complexed or co-located proteins, to give insight into the subgraph structure of these subsets. In the next chapter, we discuss the results of running the algorithm on co-complexed proteins.

The most highly connected subgraph is worth particular note here: the fact that it was so large, had such a high connectivity, and contained so many proteins known to share functions means that it is almost certainly biologically significant. No other technique has previously identified this subgraph as a module. Computational techniques alone cannot determine the exact function of this module, but it has been given to biologists who do have the tools to decipher the nature of this module.

Of the subgraphs discussed here, the 9-clique and the 9 proteins associated with the DASH complex were also identified as modules by MCODE. However, none of the modules identified by MCODE even had a significant overlap with the 16-connected subgraph, and it is unlikely that any of the traditional clustering algorithms would find it. Its edge density is not impressive, and only the high connectivity of this subgraph causes it to stand out. The same was true of the fourth MHCS as well as most of the highly connected subgraphs found in the human Y2H network.

Algorithms to look for high connectivity can be a valuable addition to the set of tools used to study protein-protein interaction networks. While our algorithm does not produce many clusters as other algorithms do, we have demonstrated here that subgraphs with high connectivities are biologically relevant, and some wouldn't be found by other methods. We believe that looking for

highly connected subgraphs in other biological networks will continue to give biologically interesting results. As data density increases, this technique will become even more valuable.

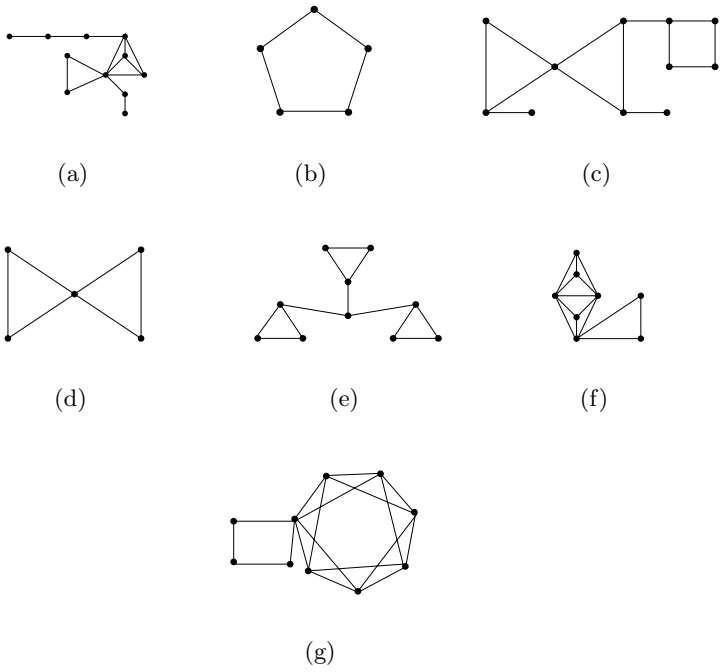


Figure 6.3: Graphs used for small-scale testing of MHCS algorithm.

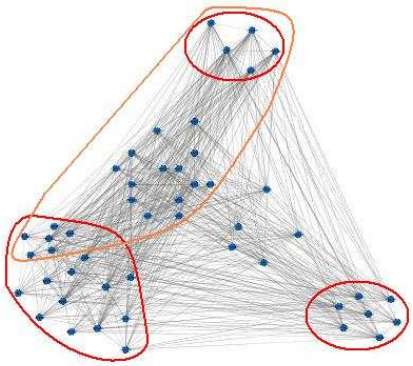


Figure 6.4: The most highly connected subgraph of the Y2H network for yeast. It is a 16- connected subgraph of 49 membrane proteins. MCODE did not find this module or any submodules when run on the entire Y2H network. Running MCODE on just this subgraph found four modules (circled), one of which overlaps two others, while the fourth is disjoint.

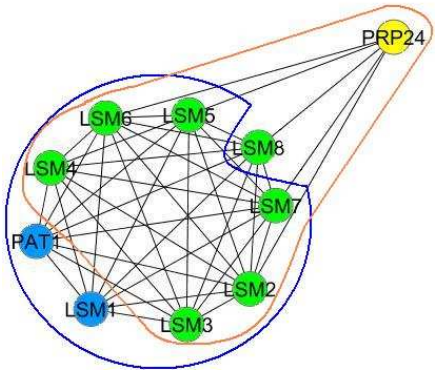


Figure 6.5: Subgraph representing the U6 snRNP and the 8-connected subgraph of 9 proteins found by our algorithm. Proteins in yellow are in the U6 snRNP, proteins in blue are in the subgraph found by the MHCS algorithm, while those in green are part of both. The proteins in the U6 snRNP are circled in orange. Also note that those proteins circled in blue, all found by the MHCS algorithm, form a separate decapping complex.

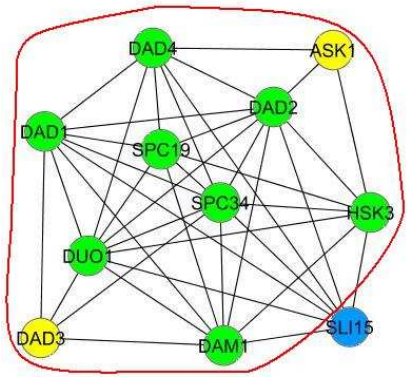


Figure 6.6: Subgraph representing the DASH complex and the 6-connected subgraph of 9 proteins found by our algorithm. Proteins in yellow are in DASH, proteins in blue are in the subgraph found by the algorithm, while those in green are part of both. The DASH complex is circled in red.

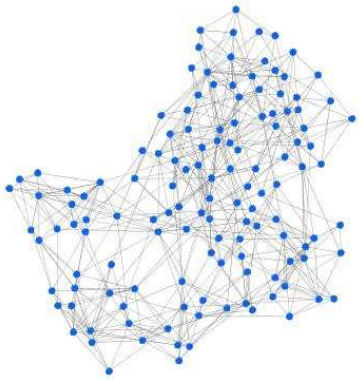


Figure 6.7: The fourth MHCS of the yeast network, which is 5-connected. The function of this subgraph is unknown.

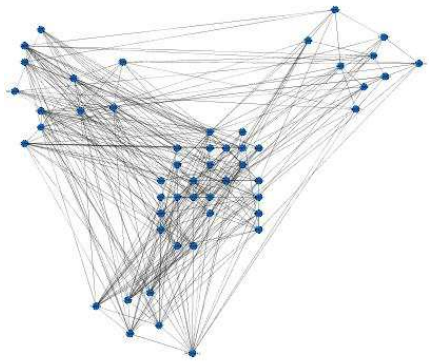


Figure 6.8: The MHCS of the human network, which is 7-connected. The function of this subgraph is unknown.

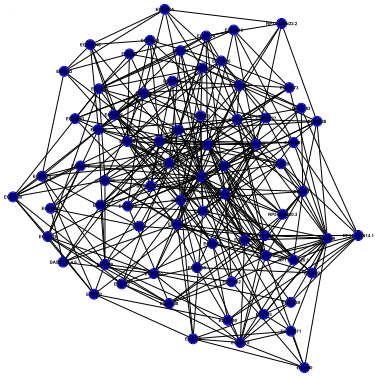


Figure 6.9: The second MHCS of the human network, which is 6-connected. The function of this subgraph is unknown.

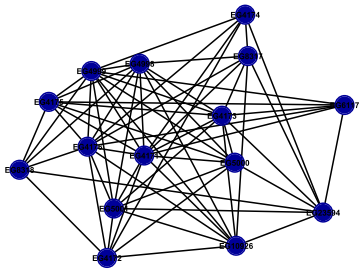


Figure 6.10: The third MHCS of the human network, which is 6-connected. Proteins in this subgraph are involved with DNA replication initiation.

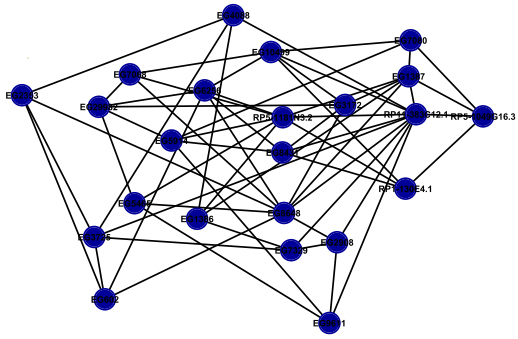


Figure 6.11: The fourth MHCS of the human network, which is 4-connected. Proteins in this subgraph are involved with transcription regulation of various sex hormones.

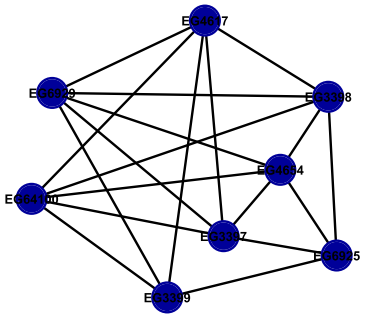


Figure 6.12: The fifth MHCS of the human network, which is 4-connected. Proteins in this subgraph are involved with transcription regulation and regulation of RNA metabolism.

Chapter 7

Connectivity and other properties in protein complexes

As discussed in Chapter 5, protein complexes are sets of proteins that bind together to perform their function, and one of the purposes of studying protein-protein interaction networks is to find these complexes. Protein complexes are one of the areas where we feel that connectivity can make a significant contribution because of the importance of the structure of complexes and stability in the complex as a whole. It is important a complex retains its structure even if a protein has a mutation that might cause it to lose an interaction (edge connectivity) or is missing altogether (vertex connectivity). Due to steric constraints, most proteins cannot interact directly with more than 4-6 proteins within a complex, so as the number of proteins in a complex increases, the edge density decreases, but the k-connectivity may remain largely constant.

There are many theories as to what a complex should look like in the interaction data. A common theory is that complexes will appear as cliques or near cliques in the interaction data. In order to test this theory and discover if connectivity might be a better indicator of complexes than edge density, we conducted a survey of topological properties for the known protein complexes in *Saccharomyces cerevisiae*. We looked at both graph connectivity and edge density as well as a number of other properties. We then compared these to the properties of random complex-like subgraphs that we called **pseudocomplexes** in order to discover the relevance of these properties.

7.1 Methods

7.1.1 Data

We looked at protein complexes in *Saccharomyces cerevisiae* from two different sources. The first source was iPFam, where we were able to obtain data about protein complexes and which proteins interact within the complex [100]. This data was obtained via x-ray crystallography, which, while not perfectly accurate, should be considered reliable. Unfortunately, only 13 complexes with at least three distinct proteins are included in this database. The second source of data on known complexes was the MIPS database [101]. The MIPS database is far more extensive, but only contains the proteins present in the complex, not the interactions that occur within the complex.

We obtained yeast two-hybrid (Y2H) interaction data from Biogrid and created an interaction graph using a composite of all Y2H studies in yeast available on Biogrid[87]. We used Y2H data rather than an affinity purification method because the Y2H data are based only on binary interactions and determine interactions without being biased towards complexes. For the same reason, we did not want to use the PCA binary interactions from Tarassov [102] because that study used complexes to filter the results. The high-throughput Y2H data, however, has a high error rate and includes both false positives (proteins that don't interact but appear to in some study) and false negatives (proteins that do interact but whose interaction has not been captured in a Y2H study).

We considered using the Y2H Union subset of interactions [103], but there aren't enough interactions in this data set between proteins in the same complex to give us meaningful results; only 25 of the 154 complexes in MIPS induced a connected graph, and of those 25, only 4 had more than 3 proteins in the data. We did not believe that this was enough data to give a meaningful picture of complexes, so we decided it was better to accept the lower quality but higher number of interactions from the composite data set. It is worthwhile to discover metrics that would allow us to find protein complexes in the abundantly available data. We therefore decided to accept a lower specificity and a higher number of false positives in order to increase the sensitivity.

In order to avoid confusion, for the rest of the chapter, we will refer to the entire collection of proteins and interactions determined by Y2H interactions as the "network." The collection of proteins and interactions in a complex will be a "graph" while a subset of those interactions will be a "subgraph."

For the complexes from iPFam, we looked at both the interactions determined by the x-ray crystallography on isolated complexes and also the graph induced in the Y2H network by the proteins determined to be in the complex. See Fig. 7.1. The x-ray crystallography data gives us an idea of how complexes might look in a complete and accurate interaction network, while the Y2H data gives us an idea of how complexes look in our real error prone data. For the complexes from MIPS, we were only able to look at the induced graphs from the Y2H data.

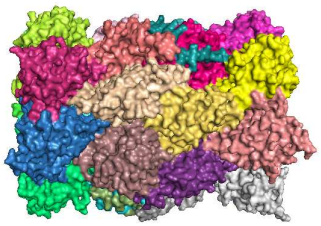
7.1.2 Graph Properties

In addition to edge density and edge and vertex connectivity, we also looked at the following statistics:

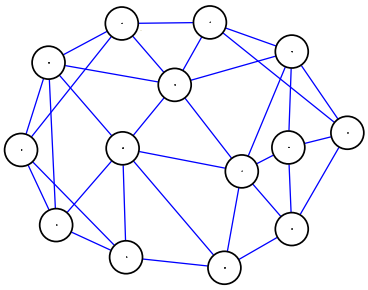
Degree Statistics: The maximum, minimum, and mean degrees for each graph, along with the standard deviation of the mean. In order to compare these statistics between complexes with differing numbers of proteins, we normalize by dividing the degree statistics by the number of vertices in the graph.

Clustering Coefficient (CC): A measure of how many of a vertex's neighbors are neighbors of each other. Over a graph or subgraph, clustering coefficient is defined as 3 times the number of triangles divided by the number of length 2 paths.

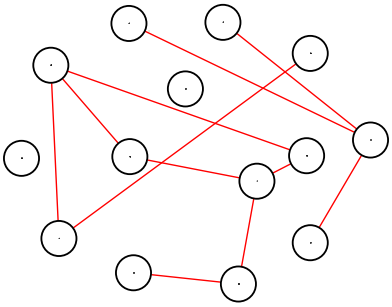
Mutual Clustering Coefficient (MCC): For a pair of vertices, the percentage of their neighbors that they share. There are several different ways of defining the mutual clustering coefficient between two vertices, but for our purposes, we define it as the number of shared neighbors divided by the minimum number of neighbors. This method was the best of the ratio methods from Goldberg and Roth [30] for assessing confidence in PPI networks. We calculate the MCC between all pairs of vertices in a complex, and as with degree, we report the maximum, minimum, mean,



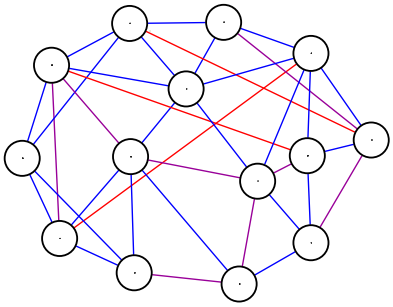
(a) The 20S proteasome, surface view.



(b) The graph constructed by looking at the proteins of the 20S proteasome and their interactions with x-ray crystallography.



(c) The subgraph of the Y2H network induced by the proteins from (b).



(d) The edges from (b) and (c). Interactions appearing only in x-ray crystallography are in blue, those appearing only in Y2H data are in red, and those appearing in both are in purple.

Figure 7.1: The 20S proteasome and the graphs that represent it. Figure (a) shows a surface image of the proteasome itself. The graph in (b) represents the interactions from the isolated complex (from iPFam), while the graph in (c) contains the same proteins but gets its edges from the Y2H network (from Biogrid).

and standard deviation.

Motifs: Particular subgraphs in each complex. We were interested in the number of triangles and 4-cycles.

Betweenness Centrality: For a vertex, the number of shortest paths between all other pairs of vertices that contain that vertex. Again, we report the maximum, minimum, mean, and standard deviation. As with the degree statistics, we normalize by dividing by the number of vertices in the graph. Because complexes are expected to be well-connected, we expect betweenness values to be small.

7.1.3 Subgraphs

For each graph of a complex, we looked at three subgraphs: 1) the original graph, which includes vertices representing all proteins in the complex; 2) a "haircut" subgraph, where we recursively eliminate all vertices of degree 1 or less, ensuring the subgraph has a minimum degree of 2 (this is the same as the haircut part of the algorithm of Bader and Hogue [49]); and 3) the most highly connected subgraph (MHCS).

The reason that we look at these additional subgraphs is that we believe that several properties will be more discernible in these subgraphs, so that these subgraphs are more likely to be able to be discovered by a complex-finding algorithm. The single vertices eliminated by the haircut are unlikely to be discovered by any complex-finding algorithm, and including them lowers the edge density, clustering coefficient, and connectivity of the graph, as well as raising the betweenness of the adjacent vertex. The MHCS is primarily designed to highlight connectivity, of course, but many other properties are also higher in the MHCS than in the original graph.

7.1.4 Assessment

In order to assess the significance of motifs in the complexes and the Y2H network as a whole, we used two different methods of generating random graphs. For the Y2H network, we generated networks with the same number of vertices and the same edge distribution by "switching." Switching works by choosing two random edges with different endpoints, (A, B) and (C, D), removing those edges, and replacing them with edges (A, D) and (C, B). We used the method recommended by Milo et al. : for a network with n vertices, the process is repeated $100n$ times to ensure proper

mixing [104]. The end result is a random network with the same degree distribution as the original network [105]. This process is repeated 10 times, giving us 10 random networks for comparison.

A somewhat different method was used to assess the significance of the properties of the complexes. Switching would only allow us to compare a protein complex graph with another graph of the same degree distribution, when what we really want is to compare it to other graphs from the Y2H network. Our question is "how likely are we to see this result in the actual network where there is not a complex?" so we seek graphs that are similar to our complexes. For each complex with at least 4 proteins, we found a "matched" graph that we call a **pseudocomplex**. A pseudocomplex P that matches a complex with n proteins is generated by taking a random triangle T from the Y2H network and letting $P_3 = T$. For $i > 3$, we generate P_i from P_{i-1} by taking a random edge in the Y2H network attached to P_{i-1} and adding the vertex at the other end and all edges from this vertex to P_{i-1}. Repeat this process until we have the same number of vertices as the original complex and let $P = P_n$. We chose a random edge rather than a random neighbor so that nodes connected by multiple edges would be more likely to be chosen, making the final graph more "complex-like." We started with a random triangle rather than a random edge for the same reason, because most (though not all) complexes contained at least one triangle. We calculated the same measures for pseudocomplexes as we did for the complex graphs, and compared our results with the real complexes.

7.2 Results

We calculated all of the statistics mentioned in the previous section for graphs generated by 13 different complexes from iPFam and 154 complexes from MIPS. Full results are in the appendices. We present some of the highlights here.

7.2.1 iPFam Complexes

There were 35 studies in iPFam that involved complexes with at least 3 proteins. Some of these studies were of the same or similar complexes; we grouped studies together if they produced

the exact same graph, i.e. the same proteins with the same set of interactions. All graphs are illustrated in Fig. 7.2 and 7.3, along with the subgraphs they induced in the Y2H data. In some cases, it is possible that two different studies of the same complex may have produced different graphs, but we will treat all distinct graphs as separate entities. This grouping gave us 13 distinct graphs. Full statistics for the complexes from iPFam are in Table 7.1; because we had interaction data from x-ray crystallography, we were able to analyze a reliable graph representation for these complexes. In all except two cases (Fig. 7.2(m) and 7.3(c)), the interactions from the x-ray crystallography produced connected graphs. In general, the edge density could be closely correlated with the number of vertices in the complex; complexes with only 3 proteins produced cliques while those with 12 or more tended to have edge densities closer to $\frac{1}{3}$. See Fig. 7.4. Most complexes were only 1-connected due to the presence of a small number of degree 1 vertices; in all cases except one, the haircut subgraphs were 2-connected. About half the complexes had a subgraph that was at least 3-connected.

When we look at the iPFam complexes in the Y2H data, we see that 9 of the 13 have all of their proteins present, 3 have slightly more than 60 percent, and 1 has only 1 out 4 proteins present. Only in one (Fig. 7.2(c-d)), a complex with 3 proteins, were all of the interactions from the x-ray crystallography present in the Y2H data. With the exception of that complex, none of the complexes induced connected graphs, and they all had edge densities of less than 0.1. In all except two cases, the haircut produced an empty subgraph. Only two complexes had a subgraph that was at least 2-connected. Comparing these results with the results obtained using the x-ray crystallography data, as is done in Table 7.1, gives us an idea of how many interactions have not been detected using a Y2H assay and how these false negatives make it difficult to detect complexes.

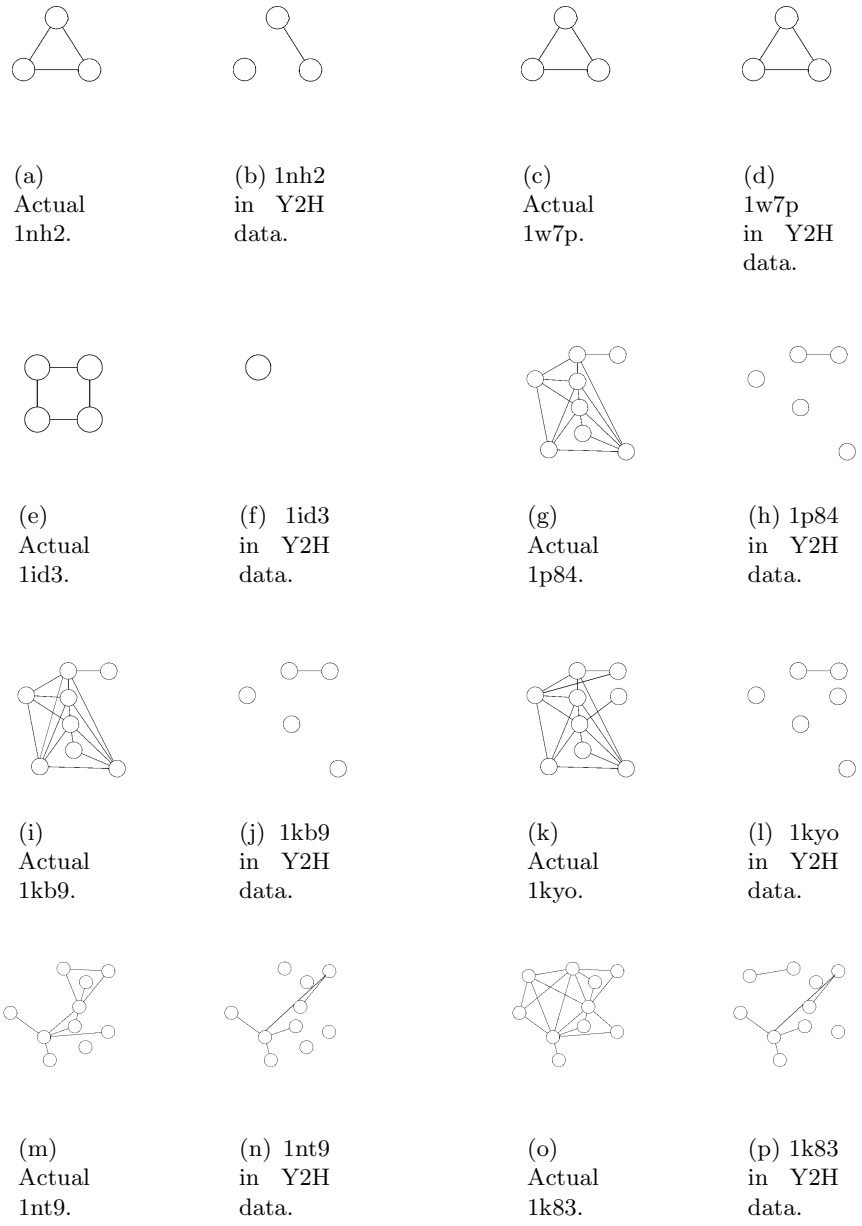
Looking at statistics other than edge density and connectivity in the iPFam complexes, we see that clustering coefficients had a similar pattern to edge density in that the value was closely correlated with the number of vertices in the complex. Mutual clustering coefficients were more scattered, but also tended to decrease as the number of vertices increased. See Fig. 7.4-7.6. In the Y2H data, most graphs had clustering coefficients of 0. Average mutual clustering coefficients were

higher, between 0.14 and 0.63.

	n		m		Edge Dens.		Max Degree	
PDB ID	x-ray	Y2H	x-ray	Y2H	x-ray	Y2H	x-ray	Y2H
1nh2	3	3	3	1(1)	1	0.33	2	1
1w7p	3	3	3	3(3)	1	1	2	2
1id3	4	1	4	0(0)	0.67	N/A	2	0
1p84	8	5	15	1(1)	0.54	0.1	5	1
1kb9	8	5	16	1(1)	0.57	0.1	5	1
1kyo	9	6	17	1(1)	0.47	0.07	6	1
1nt9	10	10	10	5(4)	0.22	0.11	5	4
1k83	10	10	18	6(5)	0.4	0.13	7	4
1sfo	10	10	19	6(5)	0.42	0.13	8	4
1pqv	12	12	11	7(5)	0.17	0.11	6	4
2b63	12	12	22	8(7)	0.33	0.12	8	4
1y1v	13	13	25	8(8)	0.32	0.10	9	4
1jd2	14	14	32	11	0.35	0.12	6	3

	CC		Ave MCC		Ave Bet.		Connect.	
PDB ID	x-ray	Y2H	x-ray	Y2H	x-ray	Y2H	x-ray	Y2H
1nh2	1	N/A	1	N/A	0	0	2	0
1w7p	1	1	1	1	0	0	2	2
1id3	0	N/A	0.33	N/A	0.5	N/A	2	N/A
1p84	0.61	N/A	0.71	N/A	2	0	1	0
1kb9	0.68	N/A	0.77	N/A	1.75	0	1	0
1kyo	0.554	N/A	0.72	N/A	2.44	0	1	0
1nt9	0.26	0	0.63	0.64	3.5	1.3	0	0
1k83	0.47	0	0.83	0.30	2.9	1.3	1	0
1sfo	0.52	0	0.89	0.30	2.8	1.3	1	0
1pqv	0.13	0	0.37	0.26	4.17	1.17	0	0
2b63	0.45	0	0.71	0.16	4.42	1.17	1	0
1y1v	0.42	0	0.66	0.16	4.85	1.08	1	0
1jd2	0.48	0	0.36	0.14	5.93	3.21	4	0

Table 7.1: Statistics for iPFam complexes. The number of proteins (n) and interactions (m), edge density (Edge Dens.), maximum degree (Max Degree), clustering coefficient (CC), average mutual clustering coefficient (Ave MCC), average betweenness (Ave Bet.), and vertex connectivity (Connect.) for each iPFam complex. The IDs given are from the Protein Data Base. x-ray = complex as determined by x-ray crystallography, Y2H = induced subgraph in yeast 2-hybrid data. The number in parentheses in the m Y2H column is the number of interactions from the x-ray crystallography that also occur in the Y2H network. "N
A" means that there were not enough vertices to calculate a given statistic.



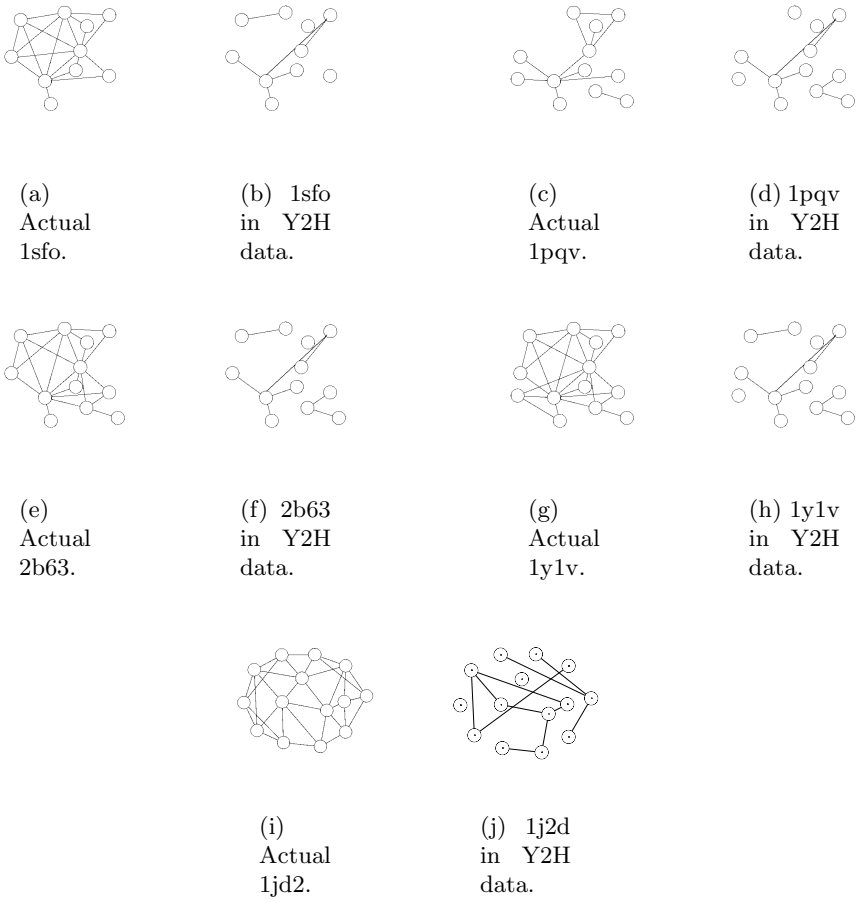
(a) Actual 1nh2. (b) 1nh2 in Y2H data. (c) Actual 1w7p. (d) 1w7p in Y2H data.

(e) Actual 1id3. (f) 1id3 in Y2H data. (g) Actual 1p84. (h) 1p84 in Y2H data.

(i) Actual 1kb9. (j) 1kb9 in Y2H data. (k) Actual 1kyo. (l) 1kyo in Y2H data.

(m) Actual 1nt9. (n) 1nt9 in Y2H data. (o) Actual 1k83. (p) 1k83 in Y2H data.

Figure 7.2: Complexes from iPFam, and those same proteins in Y2H data. IDs are from the RCSB Protein Data Base.



(a) Actual 1sfo.

(b) 1sfo in Y2H data.

(c) Actual 1pqv.

(d) 1pqv in Y2H data.

(e) Actual 2b63.

(f) 2b63 in Y2H data.

(g) Actual 1y1v.

(h) 1y1v in Y2H data.

(i) Actual 1jd2.

(j) 1j2d in Y2H data.

Figure 7.3: More complexes from iPFam and the same proteins in Y2H data. IDs are from the RCSB Protein Data Base.

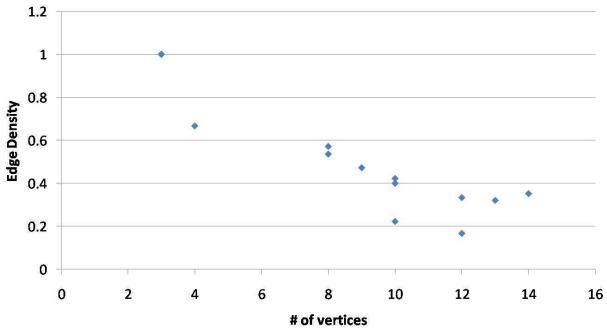


Figure 7.4: Comparison of edge density and the number of vertices in complexes from iPFam with interactions determined by x-ray crystallography.

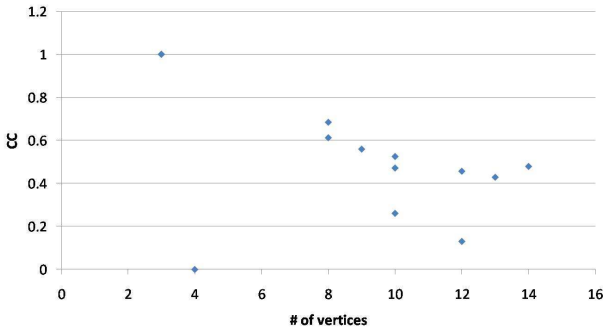


Figure 7.5: Comparison of clustering coefficient and the number of vertices in complexes from iPFam with interactions determined by x-ray crystallography.

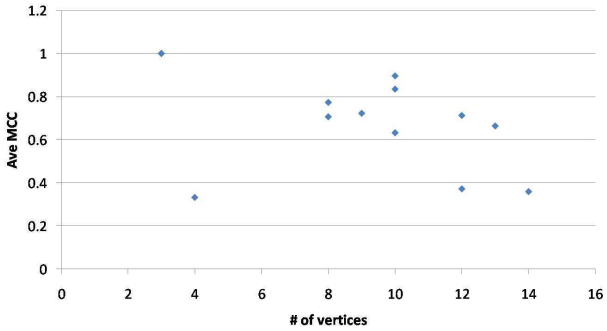


Figure 7.6: Comparison of average mutual clustering coefficient and the number of vertices in complexes from iPFam with interactions determined by x-ray crystallography.

7.2.2 MIPS Complexes

Of the 154 complexes in the MIPS database, there were 42 complexes with no edges between their component proteins in the Y2H data, 73 complexes that had interactions between the component proteins in the Y2H network but did not induce a connected graph, and 39 complexes that induced a connected graph between all the proteins that were present in the data. We report combined statistics for all 112 complexes with some interactions. For each statistic, we calculated the percentage of complexes that have values above a given threshold.

7.2.2.1 Edge Density and Connectivity

Results on edge density are in Table 7.2 and Fig. 7.7. Table 7.3 gives a comparison between edge density in real complexes and pseudocomplexes. It should be noted that the real complexes in Table 7.3 are a subset of those in Table 7.2; at least 4 proteins are needed to create a pseudocomplex.

Edge Density	Full graph (112)	Haircut (46)	MHCS (112)
1	0.08	0.46	0.5
0.9	0.08	0.46	0.52
0.8	0.1	0.59	0.55
0.7	0.11	0.63	0.59
0.6	0.25	0.76	0.85
0.5	0.33	0.85	0.9
0.4	0.36	0.89	0.95
0.2	0.63	1	0.99

Table 7.2: Fraction of complexes with edge density above a given threshold in the Y2H data. Numbers in parentheses are the number of graphs in each category.

Results on vertex connectivity for the full complexes, their haircuts, and their most highly connected subgraphs are summarized in Fig. 7.8 and Table 7.4. Virtually all complexes had the same edge connectivity as vertex connectivity, so we only give results for the stronger vertex connectivity. Comparisons with pseudocomplexes can be found in Table 7.5. Note that none of the haircut graphs had a connectivity of 1. Eliminating vertices of degree 1 is not by itself enough to guarantee that a

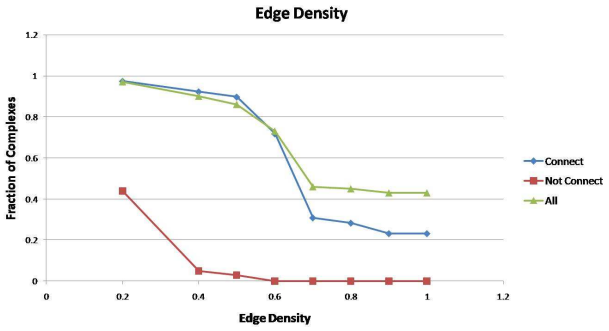


Figure 7.7: Fraction of complexes with an edge density above the given threshold. "Connected" are those complexes that induce a connected subgraph in the Y2H data and "Not connected" are the other complexes (the "Full graph" column in Table 7.2 is a combination of these 2). "All" refers to all connected components of complexes.

Edge Density	Real	Pseudo
1	.05	0
.9	.05	.05
.8	.15	.15
.7	.20	.15
.6	.45	.35
.5	.8	.65
.4	.85	.75
.2	.95	1.00

Table 7.3: Fraction of real complexes which induce a connected graph on at least 4 proteins in the data and comparable pseudocomplexes which have an edge density greater than the given threshold. Statistics based on 20 different graphs.

graph will be at least 2-connected, so this result is significant. It indicates that removing all degree 1 vertices from complexes also eliminates all articulation points, vertices whose removal disconnects the graph, leaving behind a graph where no one vertex can be removed to disconnect the graph.

7.2.2.2 Other statistics

The results for clustering coefficient and mutual clustering coefficient from MIPS are in Tables 7.6 and 7.7 and Fig. 7.9 and 7.10. As seen in Tables 7.6 and 7.7, average mutual clustering coefficient is much higher than clustering coefficient. The reason for this is that there are many

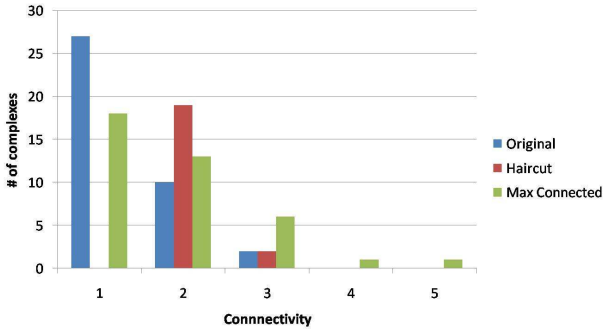


Figure 7.8: Number of complexes, their haircut graphs, and their most highly connected subgraphs with given vertex connectivity values.

Connect.	Full graph (112)	Haircut (46)	MHCS (112)
5	0	0	0.01
4	0	0	0.02
3	0.02	0.03	0.12
2	0.09	0.35	0.34

Table 7.4: Fraction of complexes with at least the given vertex connectivity value. Numbers in parentheses are the number of graphs in each category.

Connect.	Real	Pseudo
5	.05	0
4	.10	0
3	.40	.15
2	.75	1.00

Table 7.5: Fraction of real complexes which induce a connected graph on at least 4 proteins in the data and comparable pseudocomplexes which have a subgraph of at least the given vertex connectivity. Statistics based on 20 different graphs.

more 4-cycles than triangles. While triangles are overrepresented in the Y2H network as compared to a random network of the same degree distribution produced by switching (4681 v. 1609.8, 2.9 times as many), 4-cycles are even more overrepresented (98166 v. 24045.0, 4.1 times as many). The frequencies of triangles and 4-cycles relative to random networks has been calculated for a previous yeast PPI network, also with the result that both were overrepresented and 4-cycles were even more overrepresented, though this was not stated explicitly [106]. It does not, however, appear to

hold true for all PPI networks; specifically, in *Drosophila melanogaster*, triangles appear to be more overrepresented than 4-cycles [32].

In the complex graphs, 4-cycles seem to be even more overrepresented than they are in the Y2H network as a whole. While results varied, 4-cycles were overrepresented in 50% of complexes compared to pseudocomplexes, as opposed to triangles, which were only overrepresented in 29% of complexes. The overrepresentation of triangles in the real complexes was notable also, however, especially considering that all pseudocomplexes started with at least one triangle.

Comparisons between clustering coefficient and mutual clustering coefficients in real and pseudocomplexes are in Tables 7.8 and 7.9. As with edge density and connectivity, the real complexes in these tables are a subset of those in Tables 7.6 and 7.7 respectively.

CC	Full graph (84)	Haircut (46)	MHCS (84)
1	0.13	0.46	0.33
0.9	0.13	0.46	0.35
0.8	0.15	0.54	0.36
0.7	0.19	0.65	0.39
0.6	0.3	0.74	0.45
0.5	0.35	0.83	0.48
0.4	0.39	0.87	0.49
0.2	0.5	0.96	0.51

Table 7.6: Fraction of complexes with clustering coefficient above a given threshold in the Y2H data. Numbers in parentheses are the number of graphs in each category. Note that not all complexes produced a graph with a valid clustering coefficient; those that did not were omitted from the statistics.

The normalized results for maximum degree are in Table 7.10 with the comparison with pseudocomplexes in Table 7.11. In many of the complexes we looked at, there was at least one protein of high degree that had an interaction with all or almost all of the other proteins in the complex, forming a "star" or a "hub and spoke" in the graph. This has been previously suggested by Bader and Hogue as a way to model the interactions in complexes that were found experimentally using affinity-purification [26]. However, there are some problems with using this idea to search for complexes in the data. The first is that we did not notice a strong correlation between proteins

MCC	Full graph (86)	Haircut (46)	MHCS (84)
1	0.49	0.57	0.73
0.9	0.51	0.63	0.75
0.8	0.53	0.74	0.8
0.7	0.6	0.76	0.82
0.6	0.62	0.78	0.83
0.5	0.66	0.83	0.93
0.4	0.7	0.91	0.93
0.2	0.84	1	1

Table 7.7: Fraction of complexes with mutual clustering coefficient above a given threshold in the Y2H data. Numbers in parentheses are the number of graphs in each category. Note that not all complexes produced a graph with a valid mutual clustering coefficient; those that did not were omitted from the statistics.

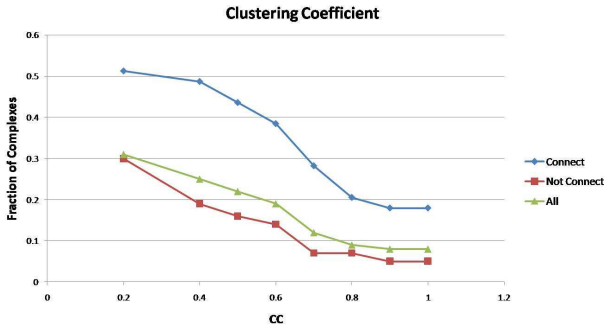


Figure 7.9: Fraction of complexes with a clustering coefficient above the given threshold. "Connected" are those complexes that induce a connected subgraph in the Y2H data and "Not connected" are the other complexes (the "Full graph" column in Table 7.6 is a combination of these 2). "All" refers to all connected components of complexes.

CC	Real	Pseudo
1	.05	0
.9	.05	0
.8	.10	.05
.7	.25	.15
.6	.45	.35
.5	.80	.55
.4	.85	.75
.2	.95	1.00

Table 7.8: Fraction of real complexes which induce a connected graph on at least 4 proteins in the data and comparable pseudocomplexes which have a clustering coefficient greater than the given threshold. Statistics based on 20 different graphs.

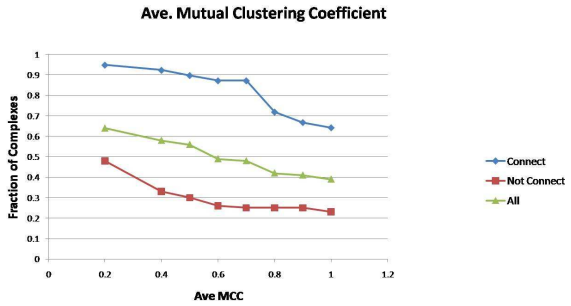


Figure 7.10: Fraction of complexes with an average mutual clustering coefficient above the given threshold. "Connected" are those complexes that induce a connected subgraph in the Y2H data and "Not connected" are the other complexes (the "Full graph" column in Table 7.7 is a combination of these 2). "All" refers to all connected components of complexes.

Ave. MCC	Real	Pseudo
1	.40	.40
.9	.45	.40
.8	.55	.65
.7	.85	.75
.6	.85	.85
.5	.90	.85
.4	.95	.90
.2	1.00	1.00

Table 7.9: Fraction of real complexes which induce a connected graph on at least 4 proteins in the data and comparable pseudocomplexes which have an average mutual clustering coefficient greater than the given threshold. Statistics based on 20 different graphs.

with high degree and proteins that appear in known complexes; roughly 30% of proteins of degree 3 or higher in our data set appeared in at least one complex, and this number remained roughly constant as we increased the degree threshold until it eventually started decreasing due to the limited number of proteins with degrees above 20. The second problem is that if we look at the protein in a complex with the most interactions with other proteins in that complex, the majority of its interactions in the Y2H data are not within the complex. Therefore, the strategy of looking for a protein of high degree and taking it and all of its neighbors as a complex seems unlikely to produce meaningful results for finding protein complexes in Y2H data.

Results for maximum betweenness are summarized in Table 7.12 with the comparison with

Max Deg. %	Full graph (112)	Haircut (46)	MHCS (112)
1	0.27	0.72	0.86
0.9	0.27	0.72	0.86
0.8	0.32	0.8	0.88
0.7	0.37	0.83	0.9
0.6	0.45	0.91	0.97
0.5	0.63	0.96	0.98
0.4	0.64	1	1
0.2	0.9	1	1

Table 7.10: Fraction of complexes with a vertex with edges to at least the given percentage of other vertices in the protein complex graph. Numbers in parentheses are the number of graphs in each category.

Max Deg %	Real	Pseudo
1	.55	.40
.9	.55	.40
.8	.75	.70
.7	.90	.85
.6	.95	.90
.5	1.00	.95
.4	1.00	1.00
.2	1.00	1.00

Table 7.11: Fraction of real complexes which induce a connected graph on at least 4 proteins in the data and comparable pseudocomplexes which have a vertex with edges to at least the given percentage of other vertices in the protein complex graph. Statistics based on 20 different graphs.

pseudocomplexes in Table 7.13. Note that for these tables, unlike the others, we report the number of complexes that were less than a given threshold rather than greater than the threshold. Some graphs did not have enough vertices (at least 3 in a connected component) to make a valid measure of betweenness; these were not included in the statistics. Betweenness statistics are not given for unconnected complexes because not all pairs of vertices have paths between them. Traditionally, betweenness has been used as a way to divide the PPI network into functional modules by identifying edges with high betweenness as edges between distinct modules or complexes, so it may seem odd that we are looking at betweenness within a complex. Our goal in looking at the betweenness statistics was to confirm that betweenness values on these nodes were low as we would expect given that we expect there to be few if any "bottleneck nodes" between any two vertices in the complex

that large numbers of shortest paths must go through. Although the minimum betweenness was almost always 0, and average betweenness was relatively small, the maximum betweenness varied quite widely, and there were some vertices with very high normalized betweenness. The maximum betweenness in fact tended to be higher in the real complexes than in the pseudocomplexes.

Max Bet. %	Full graph (37)	Haircut (21)	MHCS (37)
0	.19	.48	.32
.1	.19	.48	.51
.2	.27	.81	.57
.3	.30	.86	.57
.4	.43	1.00	.61
.5	.46	1.00	1.00
.6	.49	1.00	1.00
.8	.59	1.00	1.00

Table 7.12: Fraction of complexes for which all of their vertices have a normalized betweenness centrality less than the given threshold. Numbers in parentheses are the number of graphs in each category. Note that betweenness was calculated only for those graphs that produced a connected subgraph.

Max Bet %	Real	Pseudo
0	.05	0
.1	.05	.15
.2	.20	.30
.3	.25	.60
.4	.50	.95
.5	.55	1.00
.6	.60	1.00
.8	.80	1.00

Table 7.13: Fraction of real complexes which induce a connected graph on at least 4 proteins in the data and comparable pseudocomplexes for which all of their vertices have a normalized betweenness centrality less than the given threshold in the Y2H data. Statistics based on 20 different graphs.

7.3 Discussion

As we carried out this analysis, we were always aware of the fact that our data is error prone. We must keep in mind that the absence of an edge does not mean that there is no interaction. In order to see that we have false negatives, we need only look at the complexes with their interactions

determined by x-ray crystallography and compare them to the interactions of those same proteins in the Y2H data. Presumably, if all "real" interactions had been detected, all of the interactions that we see in the x-ray crystallography studies would be present. False positives are a more difficult matter to detect. Again, if we compare the x-ray crystallography to the Y2H data, we see edges in the Y2H graph that weren't in the x-ray crystallography. However, we cannot simply declare these false positives. It is possible that they truly are false positives. It is also possible that while "false" these interactions are significant due to the fact that they appear in the same complex (e.g. we are incorrectly labeling as a neighbor what should actually be the neighbor of a neighbor). Finally, it is possible that these are true interactions that simply do not appear as part of this complex. A recent study suggests that there are many such binary interactions and that the false positive rate for Y2H data is actually much lower than previously believed [107].

While false positives may cause problems in complex-finding algorithms, our survey suggests that false positives may be less of a problem than false negatives. If we had used a cleaner data set, we would have had fewer false positives but also fewer true positives, and we would have had even more difficult discerning complexes. Even in the data we used, complexes often did not stand out when compared to pseudocomplexes.

While the errors in the Y2H data are noteworthy, we do not feel that they represent a weakness in our study. To the contrary, a complex-finding algorithm would also be working in this same error-prone data. While it would be interesting to know how a complex would appear in an idealized network, it is more useful to know how it appears in the data we have.

7.3.1 Connectivity and Edge Density

In our examination of edge density and connectivity, we found that edge density may have been overrated as a property of complexes. We found that in Y2H data, the complexes were not particularly clique-like and edge densities were nowhere near as high as most complex-finding algorithms assumed. For example, the algorithm used by King et al. [54] looks for complexes with an edge density of at least 0.7 with a minimum number of proteins. If such a technique were applied

to Y2H binary interaction data (the data King et al. used included multiple types of interactions, some of which were not binary), our research suggests that such a technique would find all of the proteins involved in a complex for just over a tenth of known complexes with 3 or more distinct proteins (Table 7.2). An edge density threshold would find the MHCS of about 60% of known complexes, thus finding at least part of the complex, but this still leaves more than a third of complexes undetected. Also, on average, the edge densities in complexes were only slightly higher than the edge densities in the pseudocomplexes, which suggests that edge density may produce many false positives as well. Therefore, we believe that edge density has a role in developing complex-finding algorithms, but we would be skeptical of methods that purport to find complexes in Y2H data based solely on edge density.

The connectivity of complexes, on the other hand, stood out versus the connectivities of the pseudocomplexes. Our results were mixed but promising. Most complexes were only 1-connected, but this was due to a small number of degree 1 vertices. When these vertices were removed by the haircut, a 2-connected subgraph usually remained, and many complexes had 3-connected or 4-connected subgraphs. The presence of 3-connected and 4-connected subgraphs is significant; because of the way we generated our pseudocomplexes, they all had a 2-connected subgraph, but very few had a 3-connected subgraph. None of the pseudocomplexes that were designed to mimic the connected complexes had a 4-connected subgraph.

It should also be noted that while our results on connectivity in the error-prone data were promising, our results in the more accurate x-ray crystallography data were even more so. In the x-ray crystallography data, all complexes had at least a 2-connected subgraph, and the majority of complexes had a 3-connected or 4-connected subgraph. This suggests that as our data becomes more complete and accurate, highly connected subgraphs will play an even stronger role in searching for complexes.

7.3.2 Other Properties

Other graph theory properties worth mentioning in connection with protein complexes are clustering coefficient and mutual clustering coefficient. Clustering coefficient has not been as popular a parameter for complex-finding algorithms as edge density, but it has long been one of the standard tools used to study the PPI network and its subgraphs. In absolute terms, clustering coefficients were much lower than mutual clustering coefficients, but clustering coefficients in real complexes were higher than those from equivalent pseudocomplexes. Clustering coefficients were quite high in haircut graphs, but this is somewhat misleading. The haircut removes length 2 paths from the graph without removing any triangles; therefore, we would expect to increase clustering coefficient, but this increase is not necessarily significant. Mutual clustering coefficient is another statistic that has not been used extensively in complex-finding algorithms, but we believe shows promise. Many complexes had high average mutual clustering coefficients as seen in Table 7.7. It should be noted that our pseudocomplexes also have high mutual clustering coefficients (Table 7.9), but there is still reason for optimism. The average MCCs for complexes which contain some interactions between their proteins in the Y2H graph is, on average, .69, while the average over all co-complexed pairs is .36; This average is lowered by a few unconnected complexes with a large number of proteins. Another reason for believing that mutual clustering coefficient may perform well in a complex-finding algorithm is that mutual clustering coefficient considers 4-cycles as well as triangles in its calculation. As mentioned in the results section, we have found that 4-cycles are overrepresented in the Y2H network as a whole, and seem to be even more overrepresented in complexes. Both clustering coefficient and mutual clustering coefficient seem to have a correlation with complexes and would likely have a role in a new complex-finding algorithm.

7.3.3 The Role of Connectivity in Future Complex-Finding Algorithms

By itself, vertex connectivity cannot be used as the basis of a complex finding algorithm, because subgraphs with these connectivities are too common; it is easy to find 2- or 3-connected

graphs of almost any size in the PPI network. Starting with a triangle, it is possible by adding one vertex at a time to build a 3-connected subgraph of any size up to 1689 vertices. Starting with a 4-clique, it is possible to build a 3-connected graph of any size up to 913 vertices. Nevertheless, we feel these vertex-connectivity results are significant. The MHCS of graphs representing real complexes were much more highly connected than those of pseudocomplexes. The presence of a highly connected MHCS was one of the statistics that most differentiated real complexes from pseudocomplexes, suggesting that connectivity has a role in complex-finding algorithms. The absence of articulation points and the presence of highly connected subgraphs indicates something about the structure of complexes, even if k-connectivity cannot be used as the sole basis of a complex-finding algorithm.

However, connectivity could be used in conjunction with other properties in a complex-finding algorithm. Several other properties examined in this survey, most notably clustering coefficient and mutual clustering coefficient, were also highly correlated with complexes. A complex-finding algorithm based on this data could try to build a 3- or 4-connected subgraph that also had high clustering coefficients and mutual clustering coefficients. Several existing complex-finding algorithms use multiple criteria, such as MCODE (k-core, clustering coefficient, and edge density) [49], the algorithm of King et al. (clustering and edge density) [54], and the Bayesian network of Qi et al. (multiple properties, including edge density, degree statistics, and clustering coefficients) [72].

Connectivity could also be used to evaluate candidate subgraphs produced by other complex-finding algorithms. Subgraphs found by other methods could be examined to find their most highly connected subgraph, with higher confidence scores being given to those with higher connectivity values for their most highly connected subgraphs.

Finally, we hypothesize that the most highly connected subgraph of a complex graph may correspond to the "core" of a protein-complex as described by Dezso et al. [65] and Gavin et al. [66]. We plan to examine this possibility in future work. If true, this would imply that connectivity could be used in improvements to algorithms that use the core-attachment model [67, 68].

7.4 Conclusion

In order to discover whether connectivity might be helpful to discover protein complexes, we conducted a principled and comprehensive survey of how known protein complexes appear in protein-protein interaction networks. We looked at the graph induced in the Y2H network by proteins contained in a complex and measured a number of properties. We also took random "complex-like" graphs from the Y2H network and measured the same properties as we did for the complexes in order to measure the significance of our results.

We found that although the property of edge density has been by far the most common measure used when searching for complexes in the PPI network, it is not the only graph measure that is unusually high in complexes, and in fact may not be the most significant measure. The connectivity of the most highly connected subgraphs was one of the measures which was the most different in complexes from pseudocomplexes. Other measures, such as clustering coefficient and mutual clustering coefficient, show equally strong or stronger predictive power for complexes as edge density.

We hope that this data can form the basis for new, principled algorithms. We believe that algorithms based on several properties of real complexes and not solely on edge density can be a more effective way to search for complexes in interaction data.

Chapter 8

Future Work

In this chapter, we discuss some future research areas related to our work. In Section 8.1, we discuss some open problems related to our work on the LP algorithm in Chapter 3, while in Section 8.2, we discuss open bounds on special edges as discussed in Chapter 4. Section 8.3 gives an unsolved problem related to the k-ECSS problem in general, discussed in both Chapter 3 and Chapter 4. Section 8.4 gives future work related to the MHCS algorithm discussed in Chapter 6, and Section 8.5 gives future work related to our survey on protein complexes. Finally, Sections 8.6-8.8 give additional areas of research that can be applied to the work done in both Chapter 6 and Chapter 7.

8.1 Linear Programming Algorithm Bounds

8.1.1 Size of the Laminar Family

We have proved a bound on the size of a laminar family of critical sets in a k-connected simple, undirected graph that is close to $n\left(1+\frac{3}{k}+\frac{3}{k\sqrt{k}}+O(\frac{1}{k^2})\right)$. However, our lower bound example has only $n\left(1+\frac{3}{k}+\frac{1}{\sqrt{2}k\sqrt{k}}-O(\frac{1}{k^2})\right)$ sets. The upper and lower bounds are the same through the $\frac{1}{k}$ term but differ in the $\frac{1}{k\sqrt{k}}$ term, and the lower bound example has no positive $O(\frac{1}{k^2})$ term. It may be possible to close the gap between these, giving a bound that is closer to tight.

8.1.2 Further Refinement of the Rounding Method

There is also a gap between the best bound that we can prove for our improved rounding method and the best known lower bound. The approximation bound for our algorithm is $1 + \frac{1}{2k} + O(\frac{1}{k^2})$. While a careful analysis and a few modifications to the rounding method can improve the constant on the $O(\frac{1}{k^2})$ term (see Appendix A), the $\frac{1}{2k}$ term has resisted improvement. The best lower bound example we have found, however, is the trivial example shown in Fig. 8.1. This example is formed by starting with H, $(k-1)$-edge-connected graph where every vertex has degree $k-1$ (such as K_{k-1} or the example shown in Section 3.2). We take 3 copies of this graph, H, H', and H'', and connect each vertex to its "matching" vertices in the other two copies. A basic feasible solution to linear program could include all edges within H, H', and H'', and connect the matching vertices using edges of weight $\frac{1}{2}$. This example will have an approximation no better than $1 + \frac{1}{3k}$ because we will need to round at least 2 of the 3 edges connecting each v, v', and v''.

There is a significant gap between the $1 + \frac{1}{2k} + O(\frac{1}{k^2})$ upper bound and the $1 + \frac{1}{3k}$ lower bound. Further analysis could close this gap by proving a better bound on the rounding algorithm or a better lower bound example.

8.2 Bound on Special Edges for $k < 10$

8.2.1 The Case $k = 9$

As mentioned in Chapter 4, we conjecture that the $n\sigma$ bound on the number of special edges may apply in the case $k = 9$. All lemmas necessary for the proof of the $k \geq 10$ case can be extended to include $k = 9$. If $k = 9$ a share is equal to $3\frac{3}{4}(10)/2 = 18\frac{1}{2} = 2k + \frac{1}{2}$ credits, enough that a non-chain node can fulfill its obligations under conservative path payment. It is therefore plausible that we may be able to extend our proof of the $n\sigma$ bound to $k = 9$.

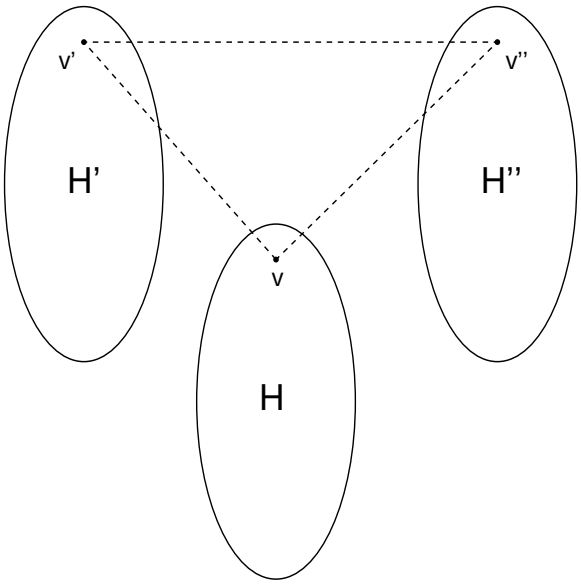


Figure 8.1: A lower bound example for any LP rounding algorithm. H is a $(k-1)$-edge-connected graph in which every vertex has degree k. Dashed edges have weight $\frac{1}{2}$. Any rounding algorithm must round 2 of the 3 edges between v, v', and v''.

8.2.2 The Case $k \leq 8$

For $k \leq 8$, there are example graphs with more than $n\sigma$ special edges [13]. For $1 \leq k \leq 3$, there are example graphs with $2n\sigma$ special edges. The $2n\sigma$ bound is tight for these values of k. For values of $4 \leq k \leq 8$, however, there is not a tight bound on the number of special edges.

8.3 Approximation Bound for the Minimum k-ECSS Problem on Simple Graphs

Currently, the best known approximation for the minimum k-ECSS problem on simple graphs is the LP rounding algorithm described in Section 3.1, with an approximation of $1+\frac{1}{2k}+O(\frac{1}{k^2})$. However, recall from Section 2.3.4 that the proof that the minimum k-ECSS problem is MAXSNP-hard on simple graphs allows for the possibility of an algorithm that achieves a $1+O(\frac{1}{k^2})$ approximation guarantee. We would like to either find a $1+O(\frac{1}{k^2})$ approximation algorithm or prove that it is NP-hard to approximate the problem to within $1+\frac{c}{k}$ for some constant c.

It should be noted that linear programing approximation methods are unlikely to provide a $1+O(\frac{1}{k^2})$ approximation algorithm for the minimum k-ECSS problem. The example in Fig. 8.1 applies not only to our algorithm but to any LP method based on rounding fractional edges. Any method of rounding the fractional edges in Fig. 8.1, no matter how clever, must round 2 of the 3 edges connecting each triple of matched vertices and thus cannot manage an approximation better than $1+\frac{1}{3k}$. We would therefore have to look at other methods to find a $1+O(\frac{1}{k^2})$ approximation.

8.4 MHCS Algorithm

8.4.1 Other Varients of the MHCS Algorithm

Other types of connectivity exist in addition to edge connectivity and vertex connectivity. Besides edge and vertex connectivity, the most commonly studied variation of connectivity is element connectivity. In this variation of the connectivity problem, we divide the vertices into terminals and nonterminals. Connectivity requirements are specified only between two terminals, and paths

between two terminals must be **element disjoint**, meaning that no two paths may travel through the same non-terminal. In some types of biological networks, it may make sense to look at element connectivity rather than edge or vertex connectivity. Therefore, we would like to modify the MHCS algorithm to look at element connectivity as well as other types of connectivity.

We would also like to modify the MHCS algorithm to look at connectivity in hypergraphs. See Section 8.8.

8.4.2 Further Analysis of Vertex Connectivity Algorithm

As mentioned in Chapter 6, there has been some work on subgraphs with high edge-connectivity. There have been algorithms similar to ours to find these subgraphs. Much less, however, has been done with vertex connectivity. There are two improvements we would like to make to our algorithm to find the most highly vertex-connected subgraph, improving the algorithm and finding other applications for it.

A closer analysis of the MHCS algorithm for vertex connectivity will most likely improve the time bound. While the connectivity of the MHCS, k_{max} affects the speed of the vertex cut algorithm, it also affects the number of recursive calls necessary. A high value for k_{max} implies a slower vertex cut algorithm but fewer recursive calls. Currently, our time bound only takes into account the first effect. We would also like to take into account the second.

Another possible area of research on the vertex connectivity algorithm is looking at other possible applications. The slicings discussed by Matula have applications not only to edge-connected subgraphs but to chromatic number as well [80]. It would be interesting to see if our similar algorithm also has applications in other areas of graph theory.

8.4.3 Subgraphs with High Connectivities

Further analysis needs to be done on the subgraphs we have found using the MHCS algorithm. Some of our subgraphs obviously correspond to a known biological module. Others, however, seem to have common biological functions but have not previously been identified as modules. We would

like to discover the signficance of these subgraphs. We would like to discover if they represent modules and if so what type of module.

Currently, collaborating biologists are working on discovering the signficance of the MHCS in yeast, the 16-connected subgraph of 49 membrane proteins. A similar analysis also could be done on some of the other yeast subgraphs as well as all of the subgraphs we found in the human Y2H data.

8.4.4 Cytoscape Plug-in

We believe that our MHCS algorithm is a valuable tool in studying PPI and other types of biological networks. We would therefore like to make an implementation of our algorithm available to the wider community. We are currently working on creating a cytoscape plug-in that will enable users of cytoscape to calculate the most highly connected subgraph of a network. We hope to implement versions of this plug-in for both edge and vertex connectivity. We also hope to implement a variant of the plug-in capable of iteratively finding multiple highly connected subgraphs.

8.5 Protein Complexes

8.5.1 Complex-finding Algorithm

We would like to use the data we have obtained in our survey of yeast protein complexes in Y2H data to build an algorithm to find unknown complexes in the data. This algorithm would most likely be based on connectivity while also taking clustering coefficient and mutual clustering coefficient into account.

8.5.2 MHCS in Complexes

In our survey of the known yeast protein complexes, we observed that many have a 3-connected or 4-connected subgraph. This property was one of those that most differentiated complexes from random subgraphs. What we have not discovered, however, is whether these subgraphs have a biological significance within the complex.

One theory is that the MHCS of a complex may be related to the complex "cores" discussed by Deszo and Gavin [65, 66]. We propose that a survey should be done to confirm or deny this theory.

8.5.3 Pseudocomplex Generation

The method of generating pseudocomplexes described in Chapter 7 may be too strict. Because not all complexes contain a triangle, we developed a second method of creating pseudocomplexes; rather than starting with a random triangle, we start with a random edge of a random triangle, then add vertices as described in the first method. By starting with an edge of a triangle, we greatly increase the chance that a given pseudocomplex will contain a triangle but it is not guaranteed. We hope to make comparisons with pseudocomplexes generated by this alternate method in the future as well as making a study of the best method for finding pseudocomplexes and other "random" subgraphs in PPI networks.

8.6 MHCS and Complexes in Other Yeast Data Sets

We ran the MHCS algorithm on protein interaction networks determined using yeast 2-hybrid (Y2H) networks. The reason that we used Y2H data is that Y2H experiments are designed to determine binary interactions. Other types of experiments, such as affinity purification (AP), give sets of proteins which may not interact directly; if two proteins A and B are both found in an affinity purification, this does not necessarily imply that A and B can bind together. For our initial experiments, therefore, we used the binary Y2H data, which gives the type of binary relationships usually implied by a graph. However, running the MHCS algorithm on AP or other data may give interesting results. The same is true of our survey of protein complexes. When looking at complexes, Y2H data has the advantage that not only is it binary, but it is not biased in favor of known complexes the way that AP data is. However, AP and other types of data are also used when looking for protein complexes. Therefore, it might be advantageous to look at complexes in this other data, though we would need to be mindful of the bias.

8.7 MHCS and Complexes in Other Organisms

Just as we would like to apply our algorithms on other data sets, we would also like to apply them to other species. Currently we have applied our MHCS algorithms to the *Saccharomyces cerevisiae* and human Y2H networks. We would like to apply it to the protein interaction networks of other species. We would like to apply it to both the Y2H network and more expansive data sets for other organisms. Likewise our survey of known protein complexes was done only for complexes from the yeast *Saccharomyces cerevisiae*. This survey could also be expanded to human protein complexes as well as the complexes from other organisms.

An expanded survey of protein complexes could also discover if the features we observed in the *Saccharomyces cerevisiae* complexes hold in other organisms. It is possible that the complexes in different organisms follow different patterns. A comparision could be made of complexes in different species in order to discover if similar features hold across all organisms.

8.8 Biological Hypernetworks

As mentioned in Section 8.6, affinity purification data does not give binary interactions. AP data gives sets of proteins which were pulled out with a bait protein. Because AP data includes sets of more than 2 proteins, it is not completely modeled by graphs. It might be better to model AP interactions using **hypergraphs**. A hypergraph is a mathematical structure consisting of vertices and hyperedges. It is similar to a graph, but unlike an edge which can only contain 2 vertices, a hyperedge can have an arbitrary number of vertices.

We would like to look at protein interaction "hypernetworks" generated by affinity purification or complex data. We would like to conduct similar studies to those we did on connectivity and complexes in PPI networks in these hypernetworks.

Bibliography

[1] Cheriyan J, Thurimella R (2000) Approximating minimum-size k-connected spanning subgraphs via matching. SIAM JOURNAL ON COMPUTING 30: 528–560.

[2] Menger K (1927) Zur allgemeinen Kurventheorie. Fund Math 10: 96–115.

[3] Garey M, Johnson D (1979) Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman.

[4] Jothi R, Raghavachari B, Varadarjan S (2003) A 5/4-approximation algorithm for minimum 2-edge-connectivity. In SODA, 725–734.

[5] Mader W (1972) Ecken vom grad n in minimalen n-fach zusammenhägenden graphen. Archive der Mathematik 23: 219–224.

[6] Mader W (1985) Minimal n-fach in minimalen n-fach zusammenhägenden digraphen. J Comb Theory B 38: 102–117.

[7] Kortsarz G, Nutov Z (2007) Approximating minimum-cost connectivity problems. In Handbook of Approximation Algorithms and Metaheuristics, CRC Press, chapter 58.

[8] Jain K (2001) A factor 2 approximation algorithm for the generalized Steiner network problem. COMBINATORICA 21: 39–60.

[9] Vazirani V (2001) Approximation Algorithms. Springer-Verlag.

[10] Goemans MX, Williamson DP (1995) A general approximation technique for constrained forest problems. SIAM Journal on Computing 24: 296–317.

[11] Gabow H (2005) On the L_∞-norm of extreme points for crossing supermodular directed network LPs. In Junger, M and Kaibel, V, editor, INTEGER PROGRAMMING AND COMBINATORIAL OPTIMIZATION, PROCEEDINGS, volume 3509 of LECTURE NOTES IN COMPUTER SCIENCE, 392–406.

[12] Gabow HN, Goemans MX, Tardos E, Williamson DP (2009) Approximating the Smallest k-Edge Connected Spanning Subgraph by LP-Rounding. NETWORKS 53: 345–357.

[13] Gabow H (2004) Special edges, and approximating the smallest directed k-connected spanning subgraph. In SODA, 227–236.

[14] Frank A (1993) Submodular Functions In Graph Theory. DISCRETE MATHEMATICS 111: 231–243.

[15] Khuller S, Raghavachari B, Young N (1995) Approximating the minimum equivalent digraph. SIAM J Comput 24: 859–872.

[16] Khuller S, Raghavachari B (1996) Improved Approximation Algorithms for Uniform Connectivity Problems. J Algorithms 21: 236–265.

[17] Gabow H (2005) An improved analysis for approximating the smallest k-edge connected spanning subgraph of a multigraph. SIAM JOURNAL ON DISCRETE MATHEMATICS 19: 1–18.

[18] Fernandes C (1998) A better approximation ratio for the minimum size k-edge-connected spanning subgraph problem. JOURNAL OF ALGORITHMS 28: 105–124.

[19] Alimonti P, Kann V (1997) Hardness of approximating problems on cubic graphs. In Bongiovanni, G and Bovet, DP and diBattista, G, editor, ALGORITHMS AND COMPLEXITY, volume 1203 of LECTURE NOTES IN COMPUTER SCIENCE, 288–298.

[20] Czumaj A, Lingas A (1999) On approximability of the minimum-cost k-connected spanning subgraph problem. In SODA, 281–290.

[21] Kortsarz G, Krauthgamer R, Lee JR (2004) Hardness of approximation for vertex-connectivity network design problems. SIAM Journal on Computing 33: 704–720.

[22] Aittokallio T, Schwikowski B (2006) Graph-based methods for analysing networks in cell biology. BRIEFINGS IN BIOINFORMATICS 7: 243–255.

[23] Shoemaker BA, Panchenko AR (2007) Deciphering protein-protein interactions. Part I. Experimental techniques and databases. PLOS COMPUTATIONAL BIOLOGY 3: 337–344.

[24] FIELDS S, SONG O (1989) A NOVEL GENETIC SYSTEM TO DETECT PROTEIN PROTEIN INTERACTIONS. NATURE 340: 245–246.

[25] Rigaut G, Shevchenko A, Rutz B, Wilm M, Mann M, et al. (1999) A generic protein purification method for protein complex characterization and proteome exploration. NATURE BIOTECHNOLOGY 17: 1030–1032.

[26] Bader G, Hogue C (2002) Analyzing yeast protein-protein interaction data obtained from different sources. NATURE BIOTECHNOLOGY 20: 991–997.

[27] Winstead ER (2002) Yeast proteomics. Genome News Network .

[28] Gibson TA, Goldberg DS (2009) Questioning the Ubiquity of Neofunctionalization. PLOS COMPUTATIONAL BIOLOGY 5.

[29] Erd˝ os P, Rényi A (1960) On the evolution of random graphs. Publications of the Mathematical Institute of the Hungarian Academy of Sciences 5: 17–61.

[30] Goldberg D, Roth F (2003) Assessing experimentally derived interactions in a small world. PROCEEDINGS OF THE NATIONAL ACADEMY OF SCIENCES OF THE UNITED STATES OF AMERICA 100: 4372–4376.

[31] Wagner A (2001) The yeast protein interaction network evolves rapidly and contains few redundant duplicate genes. MOLECULAR BIOLOGY AND EVOLUTION 18: 1283–1292.

[32] Giot L, Bader J, Brouwer C, Chaudhuri A, Kuang B, et al. (2003) A protein interaction map of Drosophila melanogaster. SCIENCE 302: 1727–1736.

[33] Yook S, Oltvai Z, Barabasi A (2004) Functional and topological characterization of protein interaction networks. PROTEOMICS 4: 928–942.

[34] Watts D, Strogatz S (1998) Collective dynamics of 'small-world' networks. NATURE 393: 440–442.

[35] Albert R, Barabasi A (2002) Statistical mechanics of complex networks. REVIEWS OF MODERN PHYSICS 74: 47–97.

[36] Albert R (2005) Scale-free networks in cell biology. JOURNAL OF CELL SCIENCE 118: 4947–4957.

[37] Schwikowski B, Uetz P, Fields S (2000) A network of proteinprotein interactions in yeast. Nature Biotechnology 18: 1257–1261.

[38] Hishigaki H, Nakai K, Ono T, Tanigami A, Takagi T (2001) Assessment of prediction accuracy of protein function from protein-protein interaction data. Yeast 18: 523–531.

[39] Chua HN, Sung WK, Wong L (2006) Exploiting indirect neighbours and topological weight to predict protein function from protein-protein interactions. Bioinformatics 22: 1623–1630.

[40] Vazquez A, Flammini A, Maritan A, Vespignani A (2003) Global protein function prediction from protein-protein interaction networks. NATURE BIOTECHNOLOGY 21: 697–700.

[41] Karaoz U, Murali T, Letovsky S, Zheng Y, Ding C, et al. (2004) Whole-genome annotation by using evidence integration in functional-linkage networks. PROCEEDINGS OF THE NATIONAL ACADEMY OF SCIENCES OF THE UNITED STATES OF AMERICA 101: 2888–2893.

[42] Nabieva E, Jim K, Agarwal A, Chazelle B, Singh M (2005) Whole-proteome prediction of protein function via graph-theoretic analysis of interaction maps. Bioinformatics 21: i302–310.

[43] Deng M, Zhang K, Mehta S, Chen T, Sun F (2003) Prediction of protein function using protein-protein interaction data. JOURNAL OF COMPUTATIONAL BIOLOGY 10: 947–960.

[44] Letovsky S, Kasif S (2003) Predicting protein function from protein/protein interaction data: a probabilistic approach. Bioinformatics 19: i197–204.

[45] Sharan R, Ulitsky I, Shamir R (2007) Network-based prediction of protein function. MOLECULAR SYSTEMS BIOLOGY 3.

[46] Hartwell L, Hopfield JJ, Leibler S, Murray AW (1999) From molecular to modular cell biology. Nature 402: C47–C52.

[47] Qi Y, Ge H (2006) Modularity and dynamics of cellular networks. PLOS COMPUTATIONAL BIOLOGY 2: 1502–1510.

[48] Schrödinger, LLC (2010) The PyMOL molecular graphics system, version 1.3r1.

[49] Bader G, Hogue C (2003) An automated method for finding molecular complexes in large protein interaction networks. BMC BIOINFORMATICS 4.

[50] Spirin V, Mirny L (2003) Protein complexes and functional modules in molecular networks. PROCEEDINGS OF THE NATIONAL ACADEMY OF SCIENCES OF THE UNITED STATES OF AMERICA 100: 12123–12128.

[51] Adamcsek B, Palla G, Farkas I, Derenyi I, Vicsek T (2006) CFinder: locating cliques and overlapping modules in biological networks. BIOINFORMATICS 22: 1021–1023.

[52] Cui G, Chen Y, Huang DS, Han K (2008) An algorithm for finding functional modules and protein complexes in protein-protein interaction networks. JOURNAL OF BIOMEDICINE AND BIOTECHNOLOGY .

[53] Bu D, Zhao Y, Cai L, Xue H, Zhu X, et al. (2003) Topological structure analysis of the protein-protein interaction network in budding yeast. NUCLEIC ACIDS RESEARCH 31: 2443–2450.

[54] King A, Przulj N, Jurisica I (2004) Protein complex prediction via cost-based clustering. BIOINFORMATICS 20: 3013–3020.

[55] Zotenko E, Guimaraes KS, Jothi R, Przytycka TM (2006) Decomposition of overlapping protein complexes: A graph theoretical method for analyzing static and dynamic protein associations. ALGORITHMS FOR MOLECULAR BIOLOGY 1.

[56] Girvan M, Newman M (2002) Community structure in social and biological networks. PROCEEDINGS OF THE NATIONAL ACADEMY OF SCIENCES OF THE UNITED STATES OF AMERICA 99: 7821–7826.

[57] Chen J, Yuan B (2006) Detecting functional modules in the yeast protein-protein interaction network. BIOINFORMATICS 22: 2283–2290.

[58] Joy M, Brock A, Ingber D, Huang S (2005) High-betweenness proteins in the yeast protein interaction network. JOURNAL OF BIOMEDICINE AND BIOTECHNOLOGY : 96–103.

[59] del Sol A, O'Meara P (2005) Small-world network approach to identify key residues in protein-protein interaction. PROTEINS-STRUCTURE FUNCTION AND BIOINFORMATICS 58: 672–682.

[60] Adarichev V, Vermes C, Hanyecz A, Mikecz K, Bremer E, et al. (2005) Gene expression profiling in murine autoimmune arthritis during the initiation and progression of joint inflammation. ARTHRITIS RESEARCH & THERAPY 7: R196–R207.

[61] Tong A, Lesage G, Bader G, Ding H, Xu H, et al. (2004) Global mapping of the yeast genetic interaction network. SCIENCE 303: 808–813.

[62] Parsons A, Brost R, Ding H, Li Z, Zhang C, et al. (2004) Integration of chemical-genetic and genetic interaction data links bioactive compounds to cellular target pathways. NATURE BIOTECHNOLOGY 22: 62–69.

[63] Rives A, Galitski T (2003) Modular organization of cellular networks. PROCEEDINGS OF THE NATIONAL ACADEMY OF SCIENCES OF THE UNITED STATES OF AMERICA 100: 1128–1133.

[64] Ravasz E, Somera A, Mongru D, Oltvai Z, Barabasi A (2002) Hierarchical organization of modularity in metabolic networks. SCIENCE 297: 1551–1555.

[65] Dezso Z, Oltvai Z, Barabasi A (2003) Bioinformatics analysis of experimentally determined protein complexes in the yeast Saccharomyces cerevisiae. GENOME RESEARCH 13: 2450–2454.

[66] Gavin A, Aloy P, Grandi P, Krause R, Boesche M, et al. (2006) Proteome survey reveals modularity of the yeast cell machinery. NATURE 440: 631–636.

[67] Leung HCM, Xiang Q, Yiu SM, Chin FYL (2009) Predicting Protein Complexes from PPI Data: A Core-Attachment Approach. JOURNAL OF COMPUTATIONAL BIOLOGY 16: 133–144.

[68] Wu M, Li X, Kwoh CK, Ng SK (2009) A core-attachment based method to detect protein complexes in PPI networks. BMC BIOINFORMATICS 10.

[69] Przulj N, Wigle D, Jurisica I (2004) Functional topology in a network of protein interactions. BIOINFORMATICS 20: 340–348.

[70] Pereira-Leal J, Enright A, Ouzounis C (2004) Detection of functional modules from protein interaction networks. PROTEINS-STRUCTURE FUNCTION AND GENETICS 54: 49–57.

[71] Rungsarityotin W, Krause R, Schodl A, Schliep A (2007) Identifying protein complexes directly from high-throughput tap data with markov random fields. BMC BIOINFORMATICS 8: 482.

[72] Qi Y, Balem F, Faloutsos C, Klein-Seetharaman J, Bar-Joseph Z (2008) Protein complex identification by supervised graph local clustering. BIOINFORMATICS 24: I250–I258.

[73] Majka J, Burgers P (2005) Function of Rad17/Mec3/Ddc1 and its partial complexes in the DNA damage checkpoint. DNA REPAIR 4: 1189–1194.

[74] Koschubs T, Seizl M, Lariviere L, Kurth F, Baumli S, et al. (2009) Identification, structure, and functional requirement of the Mediator submodule Med7N/31. EMBO JOURNAL 28: 69–80.

[75] Reeves W, Hahn S (2003) Activator-independent functions of the yeast mediator Sin4 complex in preinitiation complex formation and transcription reinitiation. MOLECULAR AND CELLULAR BIOLOGY 23: 349–358.

[76] Balciunas D, Galman C, Ronne H, Bjorklund S (1999) The Med1 subunit of the yeast mediator complex is involved in both transcriptional activation and repression. PROCEEDINGS OF THE NATIONAL ACADEMY OF SCIENCES OF THE UNITED STATES OF AMERICA 96: 376–381.

[77] Zanton SJ, Pugh BF (2006) Full and partial genome-wide assembly and disassembly of the yeast transcription machinery in response to heat shock. GENES & DEVELOPMENT 20: 2250–2265.

[78] Groll M, Koguchi Y, Huber R, Kohno J (2001) Crystal structure of the 20 S proteasome: TMC-95A complex: A non-covalent proteasome inhibitor. JOURNAL OF MOLECULAR BIOLOGY 311: 543–548.

[79] Matula DW (1969) The cohesive strength of graphs. In The Many Facets of Graph Theory (Proc. Conf., Western Mich. Univ., Kalamazoo, Mich., 1968), Springer, Berlin, 215–221.

[80] Matula DW (1972) k-components, clusters and slicings in graphs. SIAM Journal on Applied Mathematics 22: 459–480.

[81] Yuster R (2003) A note on graphs without k-connected subgraphs. ARS COMBINATORIA 67: 231–235.

[82] Brinkmeier M (2003) Communities in graphs. In Bohme, T and Heyer, G and Unger, H, editor, INNOVATIVE INTERNET COMMUNITY SYSTEMS, SPRINGER-VERLAG BERLIN, HEIDELBERGER PLATZ 3, D-14197 BERLIN, GERMANY, volume 2877 of LECTURE NOTES IN COMPUTER SCIENCE, 20–35.

[83] Hartuv E, Shamir R (2000) A clustering algorithm based on graph connectivity. INFORMATION PROCESSING LETTERS 76: 175–181.

[84] Diestel R (1997) Graph Theory, Springer. Graduate Texts in Mathematics, second edition, 50–55.

[85] GABOW H (1995) A MATROID APPROACH TO FINDING EDGE-CONNECTIVITY AND PACKING ARBORESCENCES. JOURNAL OF COMPUTER AND SYSTEM SCIENCES 50: 259–273.

[86] Gabow HN (2006) Using expander graphs to find vertex connectivity. JOURNAL OF THE ACM 53: 800–844.

[87] Stark C, Breitkreutz B, Reguly T, Boucher L, Breitkreutz A, et al. (2006) Biogrid: a general repository for interaction datasets. NUCLEIC ACIDS RES 34: D169–D172.

[88] NAGAMOCHI H, IBARAKI T (1992) COMPUTING EDGE-CONNECTIVITY IN MULTIGRAPHS AND CAPACITATED GRAPHS. SIAM JOURNAL ON DISCRETE MATHEMATICS 5: 54–66.

[89] Even S, Tarjan RE (1975) Network flow and testing graph connectivity. SIAM Journal on Computing 4: 507–518.

[90] Miller J, Lo R, Ben-Hur A, Desmarais C, Stagljar I, et al. (2005) Large-scale identification of yeast integral membrane protein interactions. PROCEEDINGS OF THE NATIONAL ACADEMY OF SCIENCES OF THE UNITED STATES OF AMERICA 102: 12123–12128.

[91] Berriz GF, Beaver JE, Cenik C, Tasan M, Roth FP (2009) Next generation software for functional trend analysis. Bioinformatics 25: 3043–3044.

[92] Chowdhury A, Tharun S (2009) Activation of decapping involves binding of the mRNA and facilitation of the post-binding steps by the Lsm1-7Pat1 complex. RNA 15: 1837–1848.

[93] Karaduman R, Dube P, Stark H, Fabrizio P, Kastner B, et al. (2005) Structure of yeast U6 snRNPs: Arrangement of Prp24p and the LSm complex as revealed by electron microscopy. RNA 14: 2528–2537.

[94] Miranda J, Wulf PD, Sorger PK, Harrison SC (2009) The yeast DASH complex forms closed rings on microtubules. Nature Structural and Molecular Biology 12: 138–143.

[95] Westermann S, Avila-Sakar A, Wang HW, Niederstrasser H, Wong J, et al. (2005) Formation of a dynamic kinetochore- microtubule interface through assembly of the dam1 ring complex. Molecular Cell 17: 277 – 290.

[96] Sandall S, Severin F, McLeod IX, Yates JR III, Oegema K, et al. (2006) A Bir1-Sli15 complex connects centromeres to microtubules and is required to sense kinetochore tension. CELL 127: 1179–1191.

[97] HAO J, ORLIN J (1994) A FASTER ALGORITHM FOR FINDING THE MINIMUM CUT IN A DIRECTED GRAPH. JOURNAL OF ALGORITHMS 17: 424–446.

[98] Karger D, Stein C (1993) An $\tilde{O}(n^2)$ algorithm for minimum cuts. In STOC 25, 757–765.

[99] Chekuri CS, Goldberg AV, Karger DR, Levine MS, Stein C (1997) Experimental study of minimum cut algorithms. In SODA '97: Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 324–333.

[100] Finn RD, Marshall M, Bateman A (2005) ipfam: visualization of protein-protein interactions in pdb at domain and amino acid resolutions. BIOINFOMATICS 21: 410–412.

[101] Mewes HW, Frishman D, Mayer KFX, Münsterkötter M, Noubibou O, et al. (2006) Mips: analysis and annotation of proteins from whole genomes in 2005. NUCLEIC ACIDS RES 34: D169–D172.

[102] Tarassov K, Messier V, Landry CR, Radinovic S, Molina MMS, et al. (2008) An in vivo map of the yeast protein interactome. SCIENCE 320: 1465–1470.

[103] Yu H, Braun P, Yildirim MA, Lemmens I, Venkatesan K, et al. (2008) High-quality binary protein interaction map of the yeast interactome network. SCIENCE 322: 104–110.

[104] Milo R, Kashtan N, Itzkovitz S, Newman M, Alon U (2004) On the uniform generation of random graphs with prescribed degree sequences, http://aps.arxiv.org/abs/cond-mat/0312028/.

[105] Roberts J (2000) Simple methods for simulating sociomatrices with given marginal totals. SOCIAL NETWORKS 22: 273–283.

[106] Przulj N, Corneil D, Jurisica I (2004) Modeling interactome: scale-free or geometric? BIOINFORMATICS 20: 3508–3515.

[107] Braun P, Tasan M, Dreze M, Barrios-Rodiles M, Lemmens I, et al. (2009) An experimentally derived confidence score for binary protein-protein interactions. NATURE METHODS 6: 91–98.

[108] Khuller S (1997) Approximation algorithms for finding highly connected subgraphs. In Approximation Algorithms for NP-hard Problems, PWS Publishing.

[109] KHULLER S, VISHKIN U (1994) BICONNECTIVITY APPROXIMATIONS AND GRAPH CARVINGS. JOURNAL OF THE ACM 41: 214–235.

[110] CAI M (1993) THE NUMBER OF VERTICES OF DEGREE-K IN A MINIMALLY K-EDGE-CONNECTED GRAPH. JOURNAL OF COMBINATORIAL THEORY SERIES B 58: 225–239.

[111] Melkonian V, Tardos T (2004) Algorithms for a network design problem with crossing supermodular demands. NETWORKS 43: 256–265.

[112] Lovász L, Plummer M (1986) Matching Theory. North-Holland Mathematics Studies 121.

Appendix A

Iterated Rounding Algorithms for the Smallest k-Edge Connected Spanning Subgraph

By Harold Gabow and Suzanne Gallagher

A preliminary version of this paper appeared in *Proc. 19th Annual ACM-SIAM Symp. on Disc. Algorithms*, 2008, pp.550–559. Full version accepted to *SICOMP* with minor revisions.

A.1 Introduction

Approximation algorithms for well-connected subgraphs are a central topic in network design (see survey papers [108, 7]). Approximating the minimum cardinality k-edge connected spanning subgraph ("smallest k-ECSS") is one of the most basic of these problems. We start by reviewing some of the past work. All graphs in this paper are undirected.

A number of authors have studied the problem for small connectivities k, especially $k = 2$ where the best known approximation ratio of Jothi et.al. is 5/4 ([4], which also reviews the history of the problem). Our focus in this paper is on general values of k. When edge weights are arbitrary nonnegative values, Khuller and Vishkin showed the minimum weight k-ECSS can be approximated to within a factor 2 [109]. This bound has resisted improvement. However the bound of 2 was extended to the more general Steiner network problem by Jain [8], who introduced the technique of iterated rounding. Our focus in this paper is on unweighted problems, mainly the smallest k-ECSS problem but also the smallest (minimum cardinality) Steiner network problem,

A factor 2 approximation for smallest k-ECSS is trivial. The best known approximation

bound for simple graphs is a combinatorial algorithm of Cheriyan and Thurimella with ratio $1 + 2/(k+1)$ [1]. For multigraphs the best known bound for a combinatorial algorithm is $< 1+\sqrt{1/e} < 1.61$[17]. Gabow et.al. showed iterated rounding achieves the bound $1 + 2/k$ ([12]; that paper also reviews history of the k-ECSS problem for general k).

Regarding hardness, Fernandes showed that finding a minimum cardinality 2-ECSS is MAX-SNP hard [18]. This was extended in [12] to show that there exists an absolute constant $c > 0$ such that for any integer $k \geq 2$, approximating the smallest k-ECSS on multigraphs to within a factor $1 + c/k$ in polynomial time implies $P = NP$.

We present the best known algorithms for approximating the smallest k-ECSS. First consider simple graphs. We begin by showing that in any simple k-edge connected graph with $k \geq 7$, any laminar family of degree k sets is smaller than the general bound: the general upper bound of $2n$ improves to $n(1 + 3/k + O(1/k\sqrt{k}))$. This immediately improves the bound for iterated rounding. The precise form of our bound on the approximation ratio, for $k \geq 7$, improves the above long-standing bound of Cheriyan and Thurimella, $1 + 2/(k + 1)$ (e.g., for $k = 7$ the bounds are $5/4$ and $17/14$ respectively, and the gap widens as k increases). We also show that our analysis is essentially tight – the bound for the laminar family is tight to within additive terms of $O(1/k\sqrt{k})$ and the bound on the approximation ratio of iterated rounding is tight to within additive terms of $O(1/k^2\sqrt{k})$.

As a second step, we present a refined version of iterated rounding for simple graphs. It improves the performance of straight iterated rounding for $k \geq 13$. Asymptotically the approximation ratio for straight iterated rounding, $1 + 1/k + O(1/k^2)$, improves to $1 + 1/2k + O(1/k^2)$.

Next consider multigraphs. We present an implementation of iterated rounding that achieves approximation ratio $1 + 21/11k < 1 + 1.91/k$ (for even k the ratio is $1 + 17/9k < 1 + 1.89/k$). Although a slight improvement of the above $1 + 2/k$ bound, this result demonstrates that the constant c in the above hardness result of [12] is < 2. Similarly it shows that the integrality gap of the natural linear program for k-ECSS is $< 1 + 1.91/k$. Our approximation ratio actually holds for the more general smallest Steiner network problem – here k denotes the average vertex demand.

Our approach, which we believe is novel, is based on the fact that the solutions to the first 2 linear programs in iterated rounding have advantageous rounding properties. These properties seem to disappear after the first two iterations. We also give an example indicating that something like our approach is necessary: No round-the-highest type algorithm (a notion which captures most known applications of iterated rounding) can improve the $1 + 2/k$ bound for k-ECSS.

The paper is organized as follows. Section 2 gives our results on laminar families for simple graphs: Sec.2.1 proves the upper bound on laminar families. (Appendix A presents the corresponding lower-bound example.) Sec.2.2 gives a simple version of our iterated rounding algorithm on simple graphs, illustrating the techniques of all the algorithms of the paper. Section 3 discusses multigraphs, and also the SN problem: Sec.3.1 gives the negative result for round-the-highest. Sec.3.2 develops our principles for rounding edges. Sec.3.3 discusses a structure amenable to rounding, "c-singleton" vertices. Sec.3.4–5 present the approximation algorithm: Sec.3.4 gives a rounding approach based on matching, which achieves our approximation ratio for even k. Sec.3.5 presents a rounding approach based on covering, which combines with the previous approach to achieve our general approximation ratio. Sec.4 gives our rounding algorithm for simple graphs, which draws on the previous ideas of rounding "singletons" as well as Sec.3.2. We conclude this section with some terminology and a review of Jain's algorithm.

We usually simplify notation by not distinguishing between an element and the singleton set formed from that element, e.g., we write $A \cup v$ rather than $A \cup \{v\}$. The given graph is $G = (V, E)$, and it has n vertices and m edges. The degree function is denoted by d, e.g., $d(S)$ denotes the degree of a set of vertices S, and $d(S, T)$ denotes the number of edges between disjoint sets of vertices S and T (so $d(S) = d(S, V - S)$). In multigraphs these expressions count each edge according to its multiplicity. If the graph is unclear it is given as a subscript, e.g., we write $d_F(S)$, where F can be just a set of edges. If x is a real-valued function on edges, a subscript of x (e.g., $d_x(S)$) indicates that we count each edge e according to its value $x(e)$. Also if F is a set of edges then $x(F)$ denotes the sum of the values $x(e)$, $e \in F$.

A graph is **k-edge connected** if every set of vertices other than $\emptyset, V$ has degree $\geq k$. This

definition extends to graphs with nonnegative real-valued edge weights. A central notion is a **critical set** – one whose degree is exactly k.

A *k-ECSS* is a k-edge connected spanning subgraph. We are interested in approximating the k-ECSS of minimum size, i.e., having the fewest number of edges possible. In the more general Steiner network problem we are given a connectivity requirement $r(v,w) \in \mathbf{Z}$ for every pair of vertices $v \neq w$. A *Steiner network (SN)* is a subgraph having at least $\geq r(v,w)$ edge disjoint paths between v and w, for all pairs of vertices. We are interested in approximating the minimum cardinality Steiner network. For any set of vertices $S \neq \emptyset, V$, let $f(S) = max\{r(v,w) : |S \cap \{v,w\}| = 1\}$. (So $f(S) = k$ for the k-ECSS problem.) The *average vertex demand* of an SN problem is $\sum_{v \in V} f(v)/n$. Special cases where we achieve better approximation ratios are the k-ECSS problem with even k and its generalization, the *even-SN problem*, where the requirement function f is even-valued.

If the desired subgraph is allowed to have nonintegral edge weights, a minimum cardinality SN corresponds to a solution to the following linear program, which can be solved in polynomial time:

$$
(LP) \qquad \begin{array}{lllll}
\text{minimize} & \sum_{e \in E} x_e & & & \\
\text{subject to} & d_x(S) & \geq & f(S) & \emptyset \subset S \subset V \\
& x_e & \leq & u_e & e \in E \\
& x_e & \geq & \ell_e & e \in E
\end{array}
$$

Variable x_e is the unknown weight of edge e. ℓ_e (u_e) is a given integral lower (upper) bound on the multiplicity of e in the solution. For simple graphs $\ell_e = 0$, $u_e = 1$.

For $I \subseteq E$, $LP(I)$ is (LP) modified to make u_e and ℓ_e equal to the same integral value for each edge $e \in I$. This notation is not precise, since it doesn't specify what the integral values are. However the values will be clear from context and can safely be omitted.

Jain proposed the method of iterated rounding to approximate the optimum integral solution to (LP) [8]. The method begins with $I = \emptyset$. Each iteration finds an extreme point solution x to $LP(I)$. Jain proves that some edge $e \notin I$ has $x_e \geq 1/2$. The algorithm adds e to I, and it fixes x_e

to the ceiling of its current value by setting both ℓ_e and u_e to that value. Then it continues with the next iteration. The algorithm halts when $I = E$.

We also need to recall the first part of the proof of correctness: Jain shows that the fractional values of x are uniquely determined as the solution to a system of linearly independent equations $d_x(S) = f(S)$, $S \in \mathcal{L}.$. Here $\mathcal{L}.$ is a laminar family of subsets of V. (A collection of sets is **laminar** if any two of its sets are either disjoint or one is contained in the other.) The number of fractional edges is exactly $|\mathcal{L}.|$.

[12] draws the following conclusion about iterated rounding for k-ECSS. Let $LP(\emptyset)$ have optimum objective value z^*. Let $\mathcal{L}.$ be the above laminar family for the extreme point solution to $LP(\emptyset)$. Then iterated rounding finds a k-ECSS containing $\leq z^* + |\mathcal{L}.|/2$ edges. (In proof, each iteration of iterated rounding increases the objective function by $\leq 1/2$, and there are $\leq |\mathcal{L}.|$ iterations, one for each fractional edge.) Any k-ECSS has $\geq \max\{z^*, kn/2\}$ edges (the latter because each vertex has degree $\geq k$). Hence iterated rounding has an approximation ratio

$$\leq 1 + |\mathcal{L}.|/kn. \tag{A.1}$$

Since $\mathcal{L}.$ is a laminar family on a ground set of n elements, its size is $\leq 2n$. This gives approximation ratio $\leq 1 + 2/k$ for iterated rounding on the k-ECSS problem.

A.2 Simple Graphs

This section begins with our upper bound for the laminar families of degree k sets in k-edge connected simple graphs. Then it presents a simplfied version of our rounding algorithm. This illustrates the techniques of this paper.

A.2.1 The Laminar Family

Recall the approximation ratio (A.1) for iterated rounding on k-ECSS. We will show that $|\mathcal{L}.|$ is small when the graph is simple. An improved approximation ratio for iterated rounding immediately follows.

The bulk of the section is an analysis of laminar families of critical sets in k-edge connected simple graphs. (The above $\mathcal{L}.$ is an example.) Actually for iterated rounding we need a slightly more general result – the graphs of interest are *weighted simple graphs*, i.e., simple graphs where every edge has a numerical weight in the interval $[0, 1]$.

Consider a k-edge connected weighted simple graph $G = (V, E)$. (As mentioned in Sec.1 the meaning of k-edge connectivity is clear for such a graph.) An *r-critical set* is a set of vertices whose complement is the disjoint union of r sets of degree k. For instance a critical set is 1-critical. And we shall see that an r-critical set models a set of $\mathcal{L}.$ that has $r - 1$ children.

An r-critical set S of cardinality s has

$$rk \geq s(k - s + 1). \tag{A.2}$$

This follows since each vertex of S has $\leq s - 1$ neighbors in S, and so it is joined to $V - S$ by edges of total weight $\geq k - (s - 1)$.

Lemma 19 (Criticality)**.** *An r-critical set has cardinality $\leq r$ or $\geq k - r + 1$ if at least one of the following conditions holds:*

i) $r = 1$,

ii) $r = 2$ *and* $k \geq 7$,

iii) $r \leq \sqrt{k} - 1/2$.

Remark Part (i) is a well-known fact about critical sets. Part (ii) appears in [13] for digraphs, where the sets are called "bicritical", and is also similar to [110, Claim 3].

Proof. The right-hand side of (A.2) is the quadratic function $f(s) = s(k - s + 1)$, which is concave down and symmetric about $(k + 1)/2$. So $f(r + 1) = f(k - r) = (r + 1)(k - r)$, and to show an r-critical set has the desired cardinality, it suffices to show $f(r + 1) = (r + 1)(k - r) > rk$. This inequality is equivalent to

$$k > r(r + 1).$$

If $r = 1$ this amounts to $k > 3$. (i) follows, since the the claim for $r = 1$ is vacuous if $k \leq 2$. It is easy to see that (ii) and (iii) follow as well. □

The key fact is a property of laminar families of critical sets in k-edge connected weighted simple graphs. It turns out that singleton sets in the family are irrelevant. So we concentrate on a laminar family $\mathcal{N}.$ of sets, each of which has cardinality > 1 and degree exactly k. ($\mathcal{N}.$ stands for "nonsingleton".) Assume $k \geq 7$ so Criticality(ii) applies.

We use tree terminology to refer to sets of $\mathcal{N}.$. So for instance the *children* of a set $S \in \mathcal{N}.$ are the maximal proper subsets of S that are $\mathcal{N}.$-sets. An $\mathcal{N}.$-set is a *leaf*, *chain node*, or *branching node* depending on whether it has 0, 1 or > 1 children, respectively.

The Criticality Lemma(i) shows a leaf S has $|S| \geq k$. Similarly, Criticality(ii) shows a chain node S with unique child C has $|S - C| \leq 2$ or $|S - C| \geq k - 1$. S is a *small chain node* in the former case ($|S - C| \leq 2$) and otherwise a *big chain node*. Note that the (unique) child of a small chain node can be a small chain node. However a chain node whose child is a small chain node, and whose (unique) grandchild is also a small chain node, must itself be a big chain node (Criticality(ii)).

Theorem 2.6. *In any k-edge connected weighted simple undirected graph, any laminar family of nonsingleton sets of degree k has cardinality $n(\frac{3}{k} + \frac{3}{k\sqrt{k}} + O(\frac{1}{k^2}))$.*

Proof: Let $\mathcal{N}.$ be a laminar family of nonsingleton sets of degree k. Assume $k \geq 7$ as before.

Say that a set $S \in \mathcal{N}.$ has *excess* Δ *over sets* $X_i,\ 1 \leq i \leq \ell$, if

$$|S - \bigcup_{i=1}^{\ell} X_i| \geq \ell + \Delta$$

where the ℓ sets X_i are pairwise disjoint descendants of S in $\mathcal{N}.$. We will concentrate on excess values $\Delta \in \{-1, 0, 1, 2\}$. Note that any set $S \in \mathcal{N}.$ has excess -1 over itself (i.e., $\ell = 1$ and $X_1 = S$).

The significance of this notion stems from the fact that excess 2 sets actually have a larger excess, because of criticality. The following claim makes this precise, and will be a key property in the proof.

Claim 1 *For $\ell \leq \sqrt{k} - 3/2$, an $\mathcal{N}.$-set with an excess of 2 actually has an excess of $k - 2\ell$, i.e., $|S - \cup X_i| \geq \ell + 2$ implies $|S - \cup X_i| \geq k - \ell$.*

Proof: If S has excess 2 over $\{X_i\}_{i=1}^{\ell}$ then $S - \cup X_i$ is an $\ell + 1$-critical set of cardinality $\geq \ell + 2$. So the claim follows from Criticality(iii). $\diamondsuit$

We will define excess numbers using a system of labels. Each $\mathcal{N}.$-set will get a label $\Delta \in \{-1, 0, 1\}$. As mentioned above, a label $\Delta = -1$ simply means the set has excess -1 over itself. Otherwise if S is a nonleaf $\mathcal{N}.$-set, let $X(S)$ denote the maximal proper descendants of S that are labelled -1. A label $\Delta \in \{0, 1\}$ means that S has excess Δ over $X(S)$. Additionally some sets with label $\Delta = -1$ are actually guaranteed to have excess 2 over $X(S)$.

We use a system of credits to bound $|\mathcal{N}.|$ as follows. We will traverse the sets $S \in \mathcal{N}.$ in a bottom-up fashion. When visiting a set S we will pay 1 credit for S. We will get these credits by charging a certain subfamily $\mathcal{C}.$ of $\mathcal{N}.$-sets 3 credits each. We will also label S with a value $\Delta \in \{-1, 0, 1\}$. Depending on Δ we may deposit credits in S. A set $S \in \mathcal{N}.$ is said to be *clear* at a given point in the traversal if the following conditions all hold:

i) Every descendant of S (including S) has been paid for.

ii) $\mathcal{C}.$ contains the sets that have been charged 3 credits. Each set $C \in \mathcal{C}.$ is labelled -1. If C is not a leaf then it has excess 2 over $X(C)$.

iii) S is labelled by some $\Delta \in \{-1, 0, 1\}$. If $\Delta \geq 0$ then S has excess Δ over $X(S)$.

iv) S holds a deposit of $1 - \Delta$ credits.

The traversal will end with all maximal $\mathcal{N}.$-sets being clear. This implies $|\mathcal{N}.| \leq 3|\mathcal{C}.|$. After describing the traversal, we will deduce the bound of the theorem by analyzing $|\mathcal{C}.|$.

The bottom-up traversal begins by clearing each leaf. The following claim indicates how this is done.

Claim 2 *Any leaf can be cleared. Any big chain node can be cleared if its child is clear. Any node*

S having excess 2 over $X(S)$ can be cleared if all its children are clear.

Proof: Suppose S is a leaf. Place S in $\mathcal{C}$. and label it -1. This gives 3 credits. One credit pays for S, and the other 2 are deposited in S. Now S is clear.

The same procedure clears a node S having excess 2 over $X(S)$, if all its children are clear.

Finally suppose S is a big chain node. By definition its unique child C has $|S - C| \geq 3$. So S has excess 2 over $X(S)$. $\diamondsuit$

The next 2 claims, applied inductively, show that every node can be cleared.

Claim 3 *A small chain node can be cleared if its unique child is clear.*

Proof: Let the small chain node S have unique child C. Let C be clear, with label Δ. S has excess $1 + \Delta$ over $X(S)$. (This follows since $|S - C| \geq 1$, and either $X(S) = X(C)$ or $\Delta = -1$ and $X(S) = \{C\}$.)

Case 1 $\Delta \leq 0$.

Label S by $1+\Delta$, a valid excess by Case 1. Use 1 credit deposited in C to pay for S. Transfer the remaining $-\Delta$ credits to S. This clears S.

Case 2 $\Delta = 1$.

S has excess 2 over $X(S)$. So Claim 2 shows S can be cleared. $\diamondsuit$

Claim 4 *A branching node can be cleared if all its children are clear.*

Proof: Let S be a branching node. Let $C_i,\ i = 1, \ldots, c$ be the children of S ($c \geq 2$).

Case 1 *2 or more children are labelled* -1.

The children of Case 1 have $\geq 2 \times 2 = 4$ credits. This is more than enough to pay for S and assign it a label of -1 with a deposit of 2. This clears S.

Case 2 *At most one child is labelled* -1.

Let C_i have label Δ_i. S has excess $\Delta = \sum_{i=1}^{c} \Delta_i$ over $X(S)$. If $\Delta \geq 2$ apply Claim 2. So suppose $\Delta \leq 1$. Label S with Δ. Case 2 implies $\Delta \geq -1$ so this label is valid. The children have a total of $\sum_{i=1}^{c} 1 - \Delta_i \geq 2 - \Delta$ credits. Use 1 credit to pay for S and deposit $1 - \Delta$ credits in S, thus clearing S. $\diamondsuit$

This completes the description of the traversal. We have shown

$$|\mathcal{N}.| \leq 3|\mathcal{C}.|.$$

We will analyze $|\mathcal{C}.|$ by charging the vertices of the graph. A vertex will get charged

$$\frac{1}{k} + \frac{1}{k(\sqrt{k} - 5/2)} = \frac{1}{k} + \frac{1}{k\sqrt{k}} + O(\frac{1}{k^2}) \tag{A.3}$$

if it is in a leaf of $\mathcal{N}.$, and if not,

$$\frac{1}{k - \sqrt{k} + 3/2} \leq \frac{1}{k} + \frac{1}{k\sqrt{k}} \ . \tag{A.4}$$

Since we will pay 1 for each set of $\mathcal{C}.$, this charging scheme implies $|\mathcal{C}.|$ is at most n times the quantity of (A.3). The theorem follows.

Claim 5 *We can pay for each set $C \in \mathcal{C}.$ by charging each vertex the quantity (A.3).*

Proof: Recall that $\mathcal{C}.$ consists of the leaves of $\mathcal{N}.$ and other sets of excess 2. We consider 3 cases.

Case 1 *C is a leaf of $\mathcal{N}.$.*

C contains $\geq k$ vertices (Criticality(i)). Hence we can pay for C by charging each of its vertices $1/k$.

Case 2 *C is an excess 2 set with $|X(C)| \leq \sqrt{k} - 3/2$.*

Claim 1 shows $|C - \cup X(C)| \geq k - \sqrt{k} + 3/2$. Hence we can pay for C by charging each vertex of $C - \cup X(C)$ the quantity of (A.4). The sets $C - \cup X(C)$ for C a nonleaf of $\mathcal{C}.$ are disjoint. Hence

k	7	8	9	100	200	400
$\nu(k)$	0.500	0.462	0.429	0.0330	0.0161	0.0079
CT	1.250	1.222	1.200	1.0198	1.0100	1.0050
new	1.214	1.183	1.159	1.0058	1.0027	1.0013

Table A.1: Values for simple graphs: $\nu(k)$ = laminar family bound, CT = bound of [1], new = our bound.

a vertex gets charged at most once. Also these sets are disjoint from the leaves of $\mathcal{N}.$, so the charge (A.4) is to nonleaf vertices.

Case 3 *C is an excess 2 set with $|X(C)| \geq \sqrt{k} - 3/2$.*

Recall that a tree with ℓ leaves, where each node x has $c(x)$ children, has $\ell - 1 = \sum c(x) - 1$. Hence for any $r > 1$, at most $(\ell - 1)/(r - 1)$ nodes have $\geq r$ children. It is easy to see this implies the number of sets C in Case 3 is $\leq \ell/(\sqrt{k} - 5/2)$, for ℓ the number of leaves in $\mathcal{N}.$. Hence we can pay for the Case 3 sets by charging each vertex in a leaf $1/(k(\sqrt{k} - 5/2))$.

Cases 1 and 3 make (A.3) the total charge to a vertex in a leaf. ♢ □

We give a slightly more precise version of the theorem's bound, as it applies to iterated rounding. Define

$$\gamma(x) = \frac{x}{(x-1)(k+2)+3}, \quad c = \lfloor \sqrt{k} - 1/2 \rfloor, \quad \nu(k) = 3\gamma(c).$$

We are only interested in these functions for $k \geq 7$, for which $\sqrt{k} > c \geq 2$. We will show that for iterated rounding, $|\mathcal{C}.| \leq \gamma(c)n$, and so $|\mathcal{N}.| \leq \nu(k)n$. To get a feel for this values of ν are given in Table A.1, e.g., $\nu(7)$ implies $\mathcal{N}.$ is half of its size in general.

First note the following properties of the function $\gamma(x)$.

Lemma 20. *These properties hold for any $k \geq 7$:*

i) $\gamma(x)$ is a decreasing function for $x \geq 1$.

ii) $\gamma(c) \geq 1/(k-c+1)$.

iii) $\frac{2}{k+1} > \frac{1+\nu(k)}{k}$.

Proof. Property (i) is trivial. To prove (iii) we start by replacing the quantity c in the right-hand side expression $\nu(k) = 3\gamma(c)$ by the smaller quantity $\sqrt{k} - 3/2$. Property (i) shows this results in a stronger inequality. Establishing the stronger inequality becomes a tedious exercise in calculus, which we omit. (Note that (iii) is a reasonable inequality since the highest order term of the left- and right-hand sides are $2/k$ and $1/k$ respectively. Also for $k = 7$ the left-hand side is $1/4$ and the right-hand side is smaller, $3/14$.)

We now prove (ii). In contrast to (iii), the floors in the definition of c are crucial for (ii), since the inequality fails for $k = 7, 13, \ldots$ if the floors are dropped.

The desired inequality amounts to $c(k-c+1) \geq (c-1)(k+2)+3$, which simplifies to $k \geq c^2 + c + 1$. Suppose this last inequality fails. Integrality gives $k \leq c^2 + c$. Thus

$$c = \lfloor \sqrt{k} - 1/2 \rfloor \leq \sqrt{c^2 + c} - 1/2.$$

But $(c + 1/2)^2 = c^2 + c + 1/4 > c^2 + c$, which implies $c + 1/2 > \sqrt{c^2 + c}$. This contradicts the displayed inequality. □

Corollary 2. *Let $\mathcal{N}.$ be the laminar family of Theorem 2.6 and assume $k \geq 7$. If every leaf of $\mathcal{N}.$ contains $> k$ vertices then $|\mathcal{N}.| \leq \nu(k)n$.*

Proof. The proof of Theorem 2.6 shows $|\mathcal{N}.| \leq 3|\mathcal{C}.|$, and Claim 5 bounds $|\mathcal{C}.|$ by n times the charge to each vertex. We refine the proof of Claim 5 so that each vertex is charged $\gamma(c)$. This implies $|\mathcal{N}.| \leq 3\gamma(c)n = \nu(k)n$ as desired. First observe that from integrality, Case 2 is defined by the inequality $|X(C)| \leq \lfloor \sqrt{k} - 3/2 \rfloor = c - 1$. Hence Case 3 has $|X(C)| \geq c$.

In Case 2, putting in the floors changes the charge to $1/(k-c+1)$. Lemma 20(ii) shows this charge is $\leq \gamma(c)$ as desired.

Consider a set C in Case 3. By definition C has excess 2, i.e., $|C - \cup X(C)| \geq |X(C)| + 2$. The proof of Case 3 shows that C can be uniquely associated with $|X(C)| - 1$ leaves of $\mathcal{N}.$.

We pay for C and its associated leaves by charging the vertices of those leaves and the vertices of $C - \cup X(C)$. The number of sets to pay for is $|X(C)|$. The number of vertices charged is $\geq (|X(C)| - 1)(k+1) + |X(C)| + 2 = (|X(C)| - 1)(k+2) + 3$. So the charge to each vertex is $\gamma(|X(C)|)$. Lemma 20(i) with $|X(C)| \geq c$ shows this charge is $\leq \gamma(c)$.

We have now paid for the sets of Case 2, and the sets of Case 3 plus their associated leaves. The remaining leaves, if any, are paid for by charging their vertices $1/(k+1)$ as in Case 1. Lemma 20(ii) shows this charge is $\leq \gamma(c)$. □

Next we apply the results to iterated rounding. Consider an execution of iterated rounding for the smallest k-ECSS of a simple undirected graph with $k \geq 7$. Take any iteration of the algorithm.

Corollary 3. *Let $\mathcal{N}.$ be the family of nonsingletons in the laminar family that determines the solution vector x of (LP). Then $|\mathcal{N}.| \leq \nu(k)n$.*

Proof. We need only establish the hypothesis of Corollary 2, i.e., every leaf of $\mathcal{N}.$ has $> k$ vertices.

Consider a leaf L with exactly k vertices. Take any vertex $x \in L$. Let $d(x)$ denote the weighted degree of x. The hypothesis implies $d(x, L) \leq k - 1$. So $d(x) \geq k$ implies $d(x, V - L) \geq 1$. Summing these last inequalities for all vertices $x \in L$ gives $d(L) \geq k$. We conclude that equality holds in all of the preceding inequalities. So every fractional edge incident to a vertex $x \in L$ is actually incident to L. This implies that the constraint on fractional edges coming from the tight LP constraint $d(L) = k$ is the sum of the constraints coming from $d(x) = k$, $x \in L$. But this violates the definition of $\mathcal{L}.$, which has all its constraints linearly independent [8]. □

We get an immediate improvement in the approximation ratio for iterated rounding:

Corollary 4. *Iterated rounding approximates the smallest k-ECSS of a simple undirected graph to within a factor of $1 + \frac{1}{k} + \frac{3}{k^2} + \frac{3}{k^2\sqrt{k}} + O(\frac{1}{k^3})$. More precisely for any $k \geq 7$ the approximation ratio is $\leq 1 + (1 + \nu(k))/k$.*

Proof. Recall from (A.1) that iterated rounding has an approximation ratio $\leq 1 + |\mathcal{L}.|/kn$. If the given graph is simple, the family $\mathcal{L}.$ consists of $\leq n$ singleton sets plus a laminar family $\mathcal{N}.$ to

which Theorem 2.6 and Corollary 3 apply. This gives the bounds of the corollary. □

Lemma 20(iii) shows that whenever the corollary's bound is valid, i.e., $k \geq 7$, it improves the approximation ratio of [1] for the smallest k-ECSS of a simple graph. Table A.1 shows the bound of [1], and the values of this new bound for $k = 7, 8, 9$.

Recall the version of iterated rounding called *round-the-highest* in [111]. It is probably the simplest version of iterated rounding. It uses the rule that each iteration rounds all the edges with the highest fractional value. The Appendix concludes by indicating how the construction extends to an example where round-the-highest comes close to the upper bound of Corollary 4. Specifically round-the-highest gets approximation ratio $1 + \frac{1}{k} + \frac{3}{k^2} + \frac{1}{\sqrt{2}k^2\sqrt{k}} - O(\frac{1}{k^3})$.

A.2.2 A Rounding Algorithm

Section A.4 presents an implementation of iterated rounding that improves our approximation ratio for k-ECSS on simple graphs with $k \geq 13$. (The entries of Table A.1 for $k = 100, 200, 400$ are for that algorithm.) This section gives a simplified version that illustrates the techniques of the algorithms in this paper.

The following terminology is used throughout the paper. Each iteration of iterated rounding finds an extreme point solution x to $LP(I)$. Fix such an x. The *weight* of an edge e is the value x_e. A *fractional edge* has weight strictly between 0 and 1. F denotes the set of all fractional edges. A fractional edge is *heavy* if its weight is $\geq 1/2$. So iterated rounding rounds heavy edges to 1.

The solution x determines a laminar family $\mathcal{L}.$ on ground set V. (We will also use the following terms on arbitrary laminar families on V.) A vertex $v \in V$ with $\{v\} \in \mathcal{L}.$ is a *singleton*; any other vertex is a *nonsingleton vertex*. A *nonsingleton set* is an $\mathcal{L}.$-set that is not a singleton. (So "nonsingletons" come in 2 flavors, but there is only one type of singleton.) $\mathcal{S}.$ denotes the set of all singletons and $\mathcal{N}.$ denotes the set of all nonsingleton sets (as in the previous section). Hence we have a partition of $\mathcal{L}.$, $\mathcal{L}. = \mathcal{S}. \cup \mathcal{N}.$.

In the lower-bound example for round-the-highest discussed in Chapter 3, most vertices are

singletons of $\mathcal{L}.$, incident to exactly 2 fractional edges. We concentrate on these vertices to improve the approximation. Towards this end partition $\mathcal{S}.$ into 2 sets:

$$S_2 = \{v : v \in \mathcal{S}.,\ d_F(v) = 2\}, \quad S^3 = \{v : v \in \mathcal{S}.,\ d_F(v) \geq 3\}.$$

(Recall $d_F(v)$ counts the number of fractional edges incident to v.) Call a heavy edge **good** if it joins 2 vertices of S_2.

The approximation algorithm is iterated rounding with one additional rule: Each iteration rounds 1 edge. That edge is chosen as a good edge if possible; if there are no good edges then the iteration rounds any heavy edge.

Theorem 2.7. *The above version of iterated rounding approximates the smallest k-ECSS of a simple graph to within a factor* $1+1/2k+O(1/k^2)$. *More precisely for any* $k \geq 7$ *the approximation ratio is* $\leq 1 + 1/2k + 5\nu(k)/k$.

Proof. Consider the first iteration in which there are no good edges. Define $\mathcal{L}.$ as the laminar family corresponding to the extreme point found in this iteration. All terms like $\mathcal{S}., S_2, F$, etc. refer to $\mathcal{L}.$.

Any good edge (v, w) that was previously rounded currently has $d_F(v), d_F(w) \leq 1$ (since the rounding destroys 1 fractional edge). Hence v and w are nonsingletons of $\mathcal{L}.$ (the total weight of the fractional edges incident to an $\mathcal{L}.$-set is a positive integer). So there are $\leq (n - |\mathcal{S}.|)/2$ such edges (v, w). Thus all rounds of our algorithm increase the objective function by a total amount $\leq (n - |\mathcal{S}.|)/4 + |\mathcal{L}.|/2 = (n + |\mathcal{S}.|)/4 + |\mathcal{N}.|/2$. We will show $|\mathcal{S}.| \leq 8|\mathcal{N}.|$. Hence the objective increases by $\leq n/4 + 5|\mathcal{N}.|/2$. The theorem follows, since the smallest k-ECSS has $\geq kn/2$ edges and $|\mathcal{N}.| \leq \nu(k)n$ (Corollary 3).

We first bound $|S^3|$. The number of fractional edges is $|F| = |\mathcal{L}.| = |\mathcal{S}.| + |\mathcal{N}.|$. Thus

$$2|S_2| + 3|S^3| \leq \sum_{v \in V} d_F(v) = 2(|\mathcal{S}.| + |\mathcal{N}.|). \tag{A.5}$$

Recalling that $|\mathcal{S}.| = |S_2| + |S^3|$, this gives $|S^3| \leq 2|\mathcal{N}.|$.

Next we bound $|S_2|$. A vertex $v \in S_2$ is incident to a heavy edge. So it is incident to a good edge if both its fractional edges lead to vertices of S_2. Since no good edge currently exists, every vertex of S_2 has a fractional edge leading to a vertex not in S_2. This gives

$$|S_2| \leq \sum_{v \notin S_2} d_F(v) = 2(|\mathcal{S}.| + |\mathcal{N}.|) - 2|S_2| = 2(|S^3| + |\mathcal{N}.|). \tag{A.6}$$

So $|S_2| \leq 6|\mathcal{N}.|$ and $|\mathcal{S}.| \leq 8|\mathcal{N}.|$ as desired. □

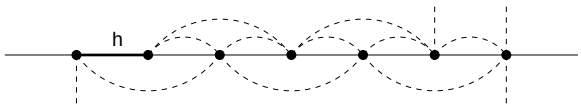
A.3 Multigraphs and Steiner Networks

This section begins with an example showing that an algorithm which can round all fractional edges in the first iteration cannot improve the worst-case bound of iterated rounding. This motivates looking at more focused rounding strategies. Section A.3.2 presents principles for such strategies, and Section A.3.3 gives the properties of "c-singleton" vertices that are amenable to rounding. The remaining sections give our approximation algorithm.

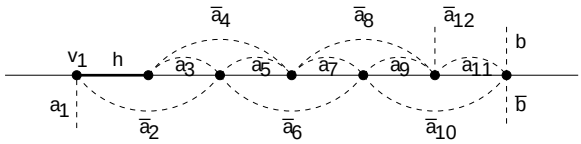
A.3.1 A Large Laminar Family

This section presents an example where round-the-highest (defined at the end of Sec.A.2.1) does not outperform its worst-case bound: For any k, we exhibit a k-ECSS problem where round-the-highest has approximation ratio that approaches $1 + 2/k$ as $n \to \infty$. The example also shows that, unlike simple graphs, the laminar family for multigraphs can be essentially as large as the general upper bound $2n$. We give two related constructions, the first for even k and the second for odd.

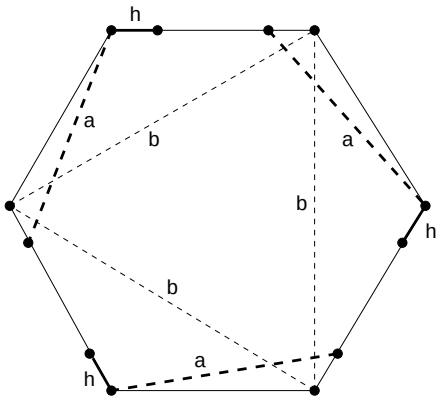
The desired example is a graph plus an optimum extreme point of (LP) that has $2n - \Theta(1)$ edges of weight $1/2$. We will present the example in the following fashion. It suffices to describe just the extreme point, say x – the graph is then implied. The requirements on x are that its fractional edge weights are uniquely determined by some set of tight constraints; x gives a k-edge connected spanning subgraph; and $x(E)$ is minimum. We will describe x and prove the first two of these properties. The last property will be obvious – every vertex v will have $d_x(v) = k$.



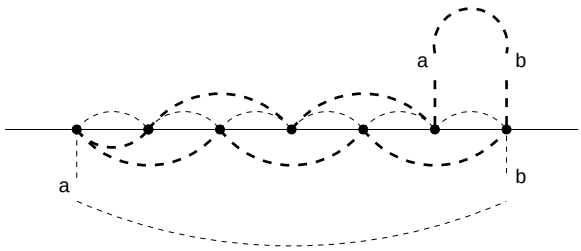
(a) Module. All but 1 solid edge weighs $h - 1$.



(b) Deducing the fractional weights



(c) Example is formed from 3 modules.



(d) Fractional cycles for a module.

Figure A.1: Bad example for iterated rounding, k even.

Even k The example for even k is based on the "module" illustrated in Fig.A.1(a). Let $h = k/2$. The module has a path of edges, drawn solid in Fig.A.1(a). The path can be arbitrarily long. The first edge of the path weighs h and the remaining edges weight $h-1$. Fig.A.1(a) also shows a weight $h-1$ edge at the beginning and at the end of the path – these edges go to other modules. The remaining edges of the module, drawn dashed, are fractional. Four fractional edges go to other modules. The remaining fractional edges form a path of "1 hop" edges and 2 paths of 2 "hop" edges.

Fig.A.1(b) shows how we deduce the weights of fractional edges. Label the vertices of the module as $v_1, v_2, \ldots$, starting with the leftmost vertex v_1 shown and proceeding rightwards. For our extreme point we require that each vertex v_i and each set $\{v_1, \ldots, v_i\}$ be tight. We will deduce that each fractional weight is either a, $\bar{a}$, b or $\bar{b}$, for some unknown values a, b. (As before the notation $\bar{a}$ is shorthand for $1-a$.) We deduce the weights advancing from left to right, in order of the weight's subscript, as follows.

Let the edge labelled a_1 weigh a. Edge $\bar{a}_2$ weighs $\bar{a}$, since v_1 is tight. Edge a_3 weighs a, since $\{v_1, v_2\}$, v_3, and $\{v_1, v_2, v_3\}$ are all tight, and the last set is the union of the first 2. Edge $\bar{a}_4$ weighs $\bar{a}$, since v_2 is tight. Now the pattern repeats starting with a_5. This continues through $\bar{a}_{12}$. The last singleton has weights b and $\bar{b}$ as shown.

Fig.A.1(c) shows the overall example, which is constructed from 3 modules. Only the solid edges of a module, its first 2 vertices, and its last 2 vertices, are shown. The top horizontal line represents one module and the 2 lower slanted lines represent the other modules. The remaining 3 lines are the weight $h-1$ edges that join modules. Each module has 2 edges labelled "a" leaving it – these correspond to the edges labelled a_1 and $\bar{a}_{12}$ in Fig.A.1(b). Similarly each module has 2 edges labelled "b" leaving it, corresponding to the edges labelled b and $\bar{b}$ in Fig.A.1(b).

We have shown that the "b" edges of each module have complementary weights. So the cycle of 3 "b" edges forces these edges to have weight 1/2. Similarly the 3 "a" edges are forced to have weight 1/2. We conclude that all fractional edges have weight 1/2 as desired.

It remains to show that the example is k-edge connected. First we will show the vertices of

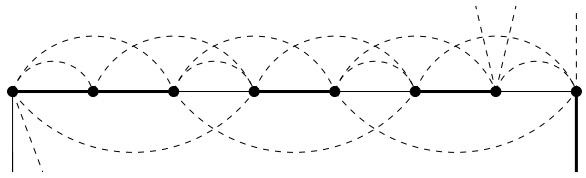
each module are k-edge connected. Once this is done, we can contract each module to 1 vertex. Fig.A.1(c) shows that we get a graph of 3 vertices joined in a cycle of weight h edges. This graph is k-edge connected, which implies the entire example is k-edge connected.

We show a module is k-edge connected by exhibiting 3 edge-disjoint cycles, each containing all the module vertices, of weights $h-1$, $1/2$ and $1/2$. The $h-1$ cycle consists of the solid edges in Fig.A.1(c). Fig.A.1(d) illustrates the two weight $1/2$ cycles. Specifically, the heavy dashed edges form one cycle. Notice that the figure indicates this cycle contains an a-b path outside the module. Similarly the light dashed edges form a cycle, containing a second a-b path outside the module. It is clear from Fig.A.1(c) that the 2 a-b paths exist and are edge-disjoint.

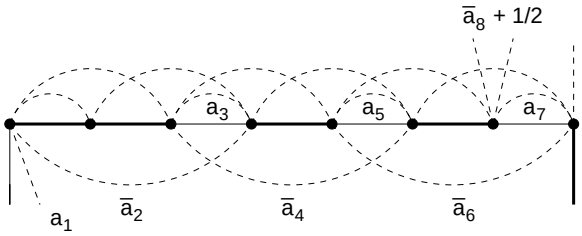
Odd k The example for odd k is illustrated in Fig.A.2. Let $h = (k-1)/2$. Fig.A.2(a) illustrates a module. As before there is a path of solid edges, with the vertical solid edges at the beginning and end going outside the module. The path can be arbitrarily long. The difference is that the weights alternate between $h-1$ and h, with the exception of the first path edge but including the 2 joining edges. As before the remaining dashed edges are fractional, with 4 going outside the module. The remaining fractional edges form 2 paths of 2 hop edges and 1 path of edges that are alternately 1 hop and 3 hop.

Fig.A.2(b) shows how we deduce the weights of fractional edges. As before we require that each vertex, as well as each set of consecutive vertices that starts with the first vertex, be tight. We begin by deducing that each unlabelled fractional edge weighs $1/2$. We will use the same principle as before: 2 disjoint sets A and B with $A \cup B = C$ and $d_x(A) = d_x(B) = d_x(C)$ have $d_x(A,B) = d_x(A, V-C) = d_x(B, V-C) = k/2$. Observe that each unlabelled fractional edge has a weight h edge at one of its ends such that the principle makes the 2 edges together weigh $k/2 = h + 1/2$. This makes the fractional edge weigh $1/2$. This applies to the (last) vertical fractional edge too.

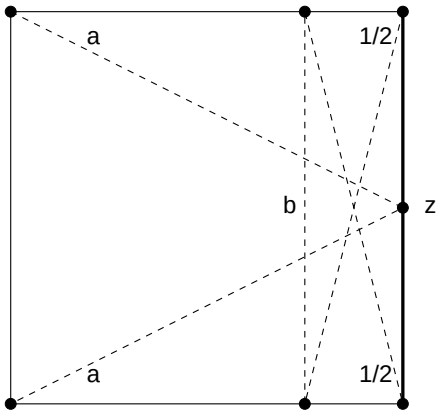
We have deduced that each vertex of the module except the penultimate vertex has 2 fractional edges weighing $1/2 + 1/2 = 1$. So the remaining fractional edges that stay inside the module



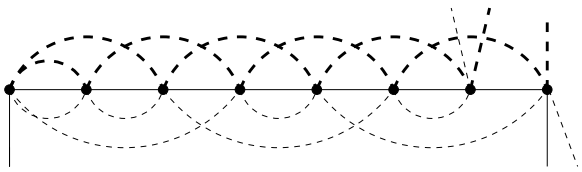
(a) Module. Solid edges weigh h (heavy) or $h-1$ (light).



(b) Deducing the fractional weights



(c) Example is formed from 2 modules and a vertex z.



(d) Fractional cycles for a module.

Figure A.2: Example for k odd.

form a path with weights alternating between a and $\bar{a}$, as shown. As for the 2 fractional edges that leave the module from the penultimate vertex, the only thing we know is that their weights sum to $k - (h + (h-1) + 1/2 + a) = 1/2 + \bar{a}$.

Fig.A.2(c) shows the overall example, which is constructed from 2 modules and a new vertex z. Only the first and last 2 vertices of each module are shown. The 2 edges labelled "a" correspond to the fractional edge leaving the first vertex of the module. Vertex z is tight. So this makes the "a" values of the two modules complementary. From the 3 other fractional edges we see that each module satisfies $\bar{a} + 1/2 = b + 1/2$. This makes the 2 "a" values identical. Hence all fractional weights – all a's and b, equal 1/2.

As before, we show the example is k-edge connected by first showing the vertices of each module are k-edge connected. Once this is done, contracting both modules to a vertex shrinks Fig.A.2(c) into a triangle of weight $h + 1/2$ edges. This graph is k-edge connected, which implies the entire example is k-edge connected.

We show a module is k-edge connected by exhibiting 4 edge-disjoint cycles, each containing all the module vertices, where 1 cycle weighs $h - 1$ and the 3 others weigh 1/2. The $h - 1$ cycle consists of the solid edges in Fig.A.2(c). A cycle of weight 1/2 consists of a path of 1 hop edges through the module, with its ends joined by 2 weight 1/2 edges incident to z. (Some of these 1 hop edges represent unused weight in h edges.) Fig.A.2(d) illustrates the two other weight 1/2 cycles. The 1 hop edges in the previous cycle are not drawn. But we draw weight 1/2 edges representing unused weight in h edges. The heavy dashed edges form one cycle. The first and last heavy edges go to the same vertex in the opposite module, as shown in Fig.A.2(c). The light dashed edges form the second cycle. Its edges alternate between 1 hop and 2 hop. Like the other cycle its first and last edges go to the same vertex.

A.3.2 Rounding Cardinality LP's

This section shows how iterated rounding can take advantage of a large collection of edges that can be rounded cheaply. The problem considered in this section is the minimum cardinality

Steiner network (SN) problem, with one exception where we explicitly specialize to the k-ECSS problem. Also the following discussion applies to both standard iterated rounding and our variant (the only difference being that our algorithm rounds a large collection of edges simultaneously).

Recall the definitions of heavy edge, the fractional edge set F, etc. from the beginning of Section A.2.2. Let I_0 denote the set of integral edges in the solution to the initial $LP(\emptyset)$. In any iteration we partition the set I (of edges fixed at the start of the iteration) into 3 sets,

$$I = I_0 \cup R \cup LR.$$

Here R is the set of all edges that the algorithm has previously rounded from a fractional weight to 1, in some iteration. LR is the set of all edges that changed from a fractional weight to 0 or 1 when some $LP(I)$, $I \neq \emptyset$, was solved. Thus F (the set of all currently fractional edges) is the set $E - I$. Δ denotes the total amount the objective value increases when we go from $LP(\emptyset)$ to the current LP solution.

At any point in the execution of the algorithm, the *lead* of the algorithm is defined to be the quantity

$$\lambda = 2n - |F| - 2\Delta.$$

To motivate this definition consider solving the k-ECSS problem using iterated rounding. First observe that if the algorithm halts with lead λ^* then the approximation ratio is $1 + 2/k - \lambda^*/kn$. In proof, the algorithm halts with $F = \emptyset$, so the total increase above optimality equals $n - \lambda^*/2$. In terms of the notation used in deriving (A.1) this says the final objective value is $\leq z^* + n - \lambda^*/2$. The rest of the calculation follows the derivation of (A.1). In fact the calculation is valid for any SN problem with average vertex demand k (here k needn't be integral). Our approach to improve the $1 + 2/k$ approximation ratio for k-ECSS is to achieve a big lead.

We return to discussing the SN problem for the rest of this section. Observe these simple properties of the lead: The algorithm has an initial nonnegative lead $2n - |F|$ right after solving $LP(\emptyset)$. Rounding the weight of a heavy edge to 1 does not decrease the lead (since $|F|$ decreases by 1 and 2Δ increases by ≤ 1). Solving $LP(I)$ changes the current solution but preserves the lead

(since immediately after rounding, the solution is feasible to $LP(I)$, so switching to an optimal solution does not increase Δ). In fact solving $LP(I)$ can increase the lead, since each edge entering LR increases the lead by 1.

Let *greedy rounding* be the version of iterated rounding where each iteration rounds every heavy edge. We analyze greedy rounding using the following terminology. Throughout the rest of the paper w denotes the "fractional weight" function, i.e., $w(e) = x_e$ if e is fractional and $w(e) = 0$ otherwise. The *bias* of a fractional edge e is $|1 - 2w(e)|$.[1] The **total bias** of a set of fractional edges S is the sum of the individual edge biases. This total bias is at least $|S - 2w(S)|$ (since $|\sum 1 - 2w(e)| \leq \sum |1 - 2w(e)|$, where both sums are over $e \in S$).

We are interested in algorithms that start off using some variant of iterated rounding and then switch to greedy rounding. Suppose we switch to greedy rounding when the set of fractional edges is F. (So F is the set of fractional edges after solving $LP(I)$ for some I, possibly even $I = \emptyset$.)

Lemma 21. *Greedy rounding solves the residual problem on F, increasing the lead by the total bias of F.*

Proof. First recall that if we start with a collection of fractional edges F having total weight $w(F)$, solving the residual problem by iterated rounding increases the objective function by $\leq w(F)$. (In proof, rounding an edge of weight w_e to 1 increases the objective by $\leq w_e$, since iterated rounding guarantees e is heavy. The solution to the next LP, x', has fractional weight $w'(F - e) \leq w(F - e)$, since after rounding e, w remains a feasible solution to the residual problem on $F - e$. So the objective function increases by $\leq w(F)$ over the entire execution of iterated rounding on F.)

Now consider greedy rounding. Divide F into the set of heavy edges H and $L = F - H$ (the "light" edges). Greedy rounding starts by rounding each $e \in H$ to 1. Rounding e increases the lead by its bias (F decreases by 1, Δ increases by $1 - w(e)$, so the lead increases by $1 - 2(1 - w(e)) = 2w(e) - 1 = |1 - 2w(e)|$). Thus we have increased the lead by the total bias of H.

We must show that the rest of the execution increases the lead by the total bias of L. Each

[1] When $w(e) < 1/2$, half this quantity is called the "halves complement of e" in [8, 9].

edge of L has bias $1 - 2w(e)$, so we need to increase the lead by $|L| - 2w(L)$.

Right after rounding H, w remains a feasible solution to the residual problem on L. So $w(L)$ does not increase when we solve the residual LP. Thus the opening remark implies iterated rounding completes the solution of the problem, increasing the objective function by $\leq w(L)$. So it increases the lead by $\geq |L| - 2w(L)$ (the number of fractional edges decreases by $|L|$ and Δ increases by $\leq w(L)$). □

Our algorithm will find edge sets with large bias by looking at vertices, so we need a vertex-based version of this fact. Recall (from Section 1) that we have degree functions d_F and d_w, as well as d_x. The *bias* of a vertex v is defined to be the quantity $|d_F(v) - 2d_w(v)|$. This quantity is a lower bound on the total bias of all fractional edges incident to v. The *total bias* of a set of vertices S is the sum of the bias of each vertex in S.

Lemma 22. *The total bias of F is at least half the total bias of any set of vertices.*

Proof. Let S be a given set of vertices, with total bias B. Let H be the set of heavy edges incident to at least 1 vertex of S. Define L similarly using the "light" edges (i.e., the fractional edges of weight $< 1/2$). The edges of H have total bias $2w(H) - |H|$, and the edges of L have total bias $|L| - 2w(L)$,

We have already noted that each vertex of S has bias at most the sum of the biases of its incident fractional edges. So

$$B \leq 2(2w(H) - |H| + |L| - 2w(L))$$

since an edge can be incident to 2 vertices of S. The last expression is twice the total bias of $H \cup L$, so the lemma follows. □

The two lemmas give a principle that we will use: Greedy rounding solves the residual problem, increasing the lead by half the total bias of any set of vertices. However this principle requires that we solve (LP) to completion. We next show how to avoid this. Specifically suppose we have a vector x that is feasible for (LP). (In the algorithm x will be an extreme point solution

that has some edges rounded up.) We want to move to an optimum solution to $LP(I)$ (I is the set of integral edges of x). Furthermore we have a set B of vertices v with $d_x(v) = f(v)$ and bias $|d_F(v) - 2d_w(v)|$ that is a positive integer. We can essentially preserve this bias in the new optimum solution, as follows.

Define the "rounding" linear program (LPR) to be (LP) with this additional constraint:

$$d_x(v) = f(v) \qquad v \in B$$

Also define the problem $LPR(I)$ to be the variant of (LPR) constructed just like $LP(I)$.

Let x have integral edges I, and let y be an optimal extreme point for $LPR(I)$. Our algorithm will move from x to y. y needn't be optimal for $LP(I)$. However our goal is only to increase the lead, and the following properties of y are the only ones needed:

Lemma 23. *i) Moving from x to y preserves the lead.*

ii) A vertex $v \in B$ has its bias preserved unless it is on an LR edge of y.

iii) As in iterated rounding, the fractional edge weights of y are uniquely determined by a system of equations corresponding to a laminar family.

Proof. i) The objective value for y is no more than the objective of x. This holds since x is feasible to $LPR(I)$. This property ensures that Δ does not increase when we switch from x to y. Hence the lead is preserved.

ii) Let F be the set of fractional edges of x. The edges of F incident to v have the same total weight in y as in x. This holds since the constraints for $LPR(I)$ preserve the value of $d_x(v)$ as well as the values $x_e,\ e \in I$. Hence if each edge of F incident to v remains fractional in y, the bias $|d_F(v) - 2d_w(v)|$ is the same in y as in x. So part (ii) holds.

iii) Any new rounding constraint defining y corresponds to an $\mathcal{L}$.-set that is a singleton, which is consistent with any laminar family. The remaining constraints defining y come from (LP). They can be made into a laminar family by exactly the same uncrossing argument as ordinary iterated rounding [8]. □

A.3.3 c-Singletons

This section continues to discuss the SN problem. We begin with a well-known property of critical sets for k-edge connectivity (Corollary 5 below), generalized to SN in the next lemma. Consider any feasible solution x of (LP). Call $S \subseteq V$ *tight* if $d_x(S) = f(S)$.

Lemma 24. *Let A, B and C be tight sets, with C the disjoint union of A and B. The quantities $d_x(A,B), d_x(A,V-C), d_x(B,V-C)$ are all half-integral. In fact they are all integral if $f(A) + f(B) + f(C)$ is even.*

Proof. By definition

$$\begin{aligned} d_x(A) &= d_x(A,V-C) + d_x(A,B), \\ d_x(B) &= d_x(B,V-C) + d_x(A,B), \\ d_x(C) &= d_x(A,V-C) + d_x(B,V-C). \end{aligned}$$

Combining these equations and using tightness shows $f(A) + f(B) - f(C) = 2d_x(A,B)$. Thus $d_x(A,B)$ is half-integral. The remaining properties claimed in the lemma follow easily. □

The proof implies this well-known fact, which will be used in the next section.

Corollary 5. *In a k-ECSS problem, $d_x(A,B) = d_x(A,V-C) = d_x(B,V-C) = k/2$.* □

We use a tree model of a laminar family that differs slightly from that of Section A.2.1. Consider a laminar family $\mathcal{L}.$ on ground set V; for convenience assume $\{V\} \notin \mathcal{L}.$. Recall the definitions of singleton, nonsingleton, etc. from the beginning of Section A.2.2. We represent $\mathcal{L}.$ as a tree $T(\mathcal{L}.)$ whose nodes correspond to the $\mathcal{L}.$-sets, plus the nonsingleton vertices, plus the set V. V is the root of $T(\mathcal{L}.)$. The parent of an $\mathcal{L}.$-set or a nonsingleton vertex is the smallest set of $\mathcal{L}. \cup \{V\}$ that properly contains it. All tree terminology for $\mathcal{L}.$ refers to $T(\mathcal{L}.)$.

A vertex of V is a *c-singleton* if it is a singleton and it has a unique sibling in $T(\mathcal{L}.)$, which is itself an $\mathcal{L}.$-set. The "c" in c-singleton stands for "complementary" – in the sense that a c-singleton

v with parent P and sibling S satisfies $P = S \cup \{v\}$ with all 3 sets belonging to $\mathcal{L}. \cup \{V\}$, so v is "complementary" to S in the laminar family.

The following properties provide edges that are amenable to rounding.

Lemma 25. *Let $\mathcal{L}.$ be the laminar family of an extreme point of (LP). Let v be a singleton.*

i) The bias of v is integral. This bias is ≥ 1 if $d_F(v)$ is odd.

ii) Suppose v has bias 0. Then v is on a heavy edge, and v is on ≥ 2 heavy edges if it is a c-singleton.

iii) $d_F(v) > 1$. In the even-SN problem if v is a c-singleton then $d_F(v) \geq 4$.

Proof. We remark that the argument does not depend on v being a single vertex – it can be an $\mathcal{L}.$-set instead of a singleton, and an appropriately defined "c-set" instead of a c-singleton.

i) A singleton v has $d_w(v)$ integral. So its bias $|d_F(v) - 2d_w(v)|$ is integral and part (i) follows.

ii) The first part is simple: If v is not on a heavy edge, i.e., each incident fractional edge weighs $< 1/2$, then $d_w(v) < d_F(v)/2$ and the bias is $d_F(v) - 2d_w(v) > 0$.

For the second part assume v is a c-singleton. Let v have parent P and sibling S in $T(\mathcal{L}.)$, so $P = S \cup \{v\}$. P is an $\mathcal{L}.$-set, since $P = V$ implies the constraints for $d_w(S)$ and $d_w(v)$ are identical, contradicting their linear independence.

Partition the edges incident to v into 2 sets: ES (EP) contains the edges joining v to S ($V - P$), respectively. Let FS (FP) denote the fractional edges of ES (EP), respectively. $FS \neq \emptyset$, since otherwise the constraint on $d_w(P)$ is the sum of the constraints for S and v, violating linear independence. Similarly $FP \neq \emptyset$.

Lemma 24 shows $x(ES)$ and $x(EP)$ are half-integral. So $w(FS)$ and $w(FP)$ are also half-integral.

We will show that if FS has no heavy edges then FP has ≥ 2 heavy edges. Since FS and FP are symmetric this implies part (ii).

The hypothesis on FS implies $2w(FS) < |FS|$. The hypothesis of part (ii) implies $|FS \cup FP| = 2w(FS \cup FP)$. So $2w(FP) > |FP|$. Half-integrality makes $2w(FP) \geq |FP| + 1$. If FP has

≤ 1 heavy edge then $2w(FP) < 2(1 + (|FP| - 1)(1/2)) = |FP| + 1$, a contradiction.

iii) A singleton has $d_w(v)$ integral, so it must be on ≥ 2 fractional edges. When v is a c-singleton and f is even, Lemma 24 shows $x(FS)$ and $x(FP)$ are integral. So $w(FS)$ and $w(FP)$ are both positive integers, making $|FS|, |FP| \geq 2$ (positivity follows from the proof of (ii)). □

We are interested in laminar families that are close to full, i.e., they contains close to $2n$ sets. Clearly most vertices are singletons. The next lemma shows there are also many c-singletons. The lemma holds for an arbitrary laminar family on ground set V.

Lemma 26. *A laminar family having $2n - \delta$ sets has $\geq n - 3\delta$ c-singletons.*

Proof. We will count the number of vertices that are not c-singletons. We can classify each vertex $v \in V$ as having 1 of 4 types:

v is a nonsingleton;

v is a singleton having a unique sibling in $T(\mathcal{L}.)$, which is itself a nonsingleton;

v is a singleton having a unique sibling in $T(\mathcal{L}.)$, which is itself an $\mathcal{L}.$-set, i.e., v is a c-singleton;

v is a singleton having > 1 sibling in $T(\mathcal{L}.)$.

Define

$$\begin{aligned} ns &= \text{the number of nonsingleton vertices,} \\ c_i &= \text{the number of nodes of } T(\mathcal{L}.) \text{ with exactly } i \text{ children } (i \geq 0). \end{aligned}$$

We get that the number of vertices that are not c-singletons is at most

$$2ns + \sum_{i \geq 3} ic_i. \tag{A.7}$$

The hypothesis defining δ shows

$$ns + \sum_{i \geq 3} (i-2)c_i \leq \delta$$

since any laminar family has $\leq 2n$ sets, each nonsingleton reduces this bound by 1, and each set with i children reduces the bound by $i - 2$. Tripling this inequality and using the fact that $i \geq 3$

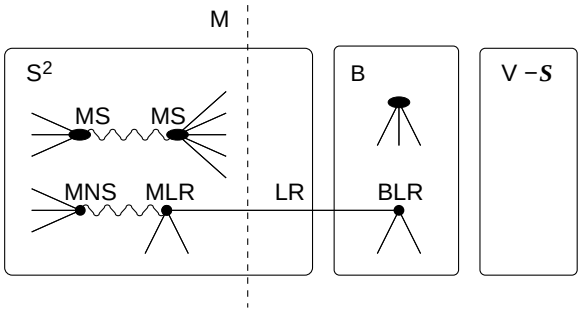


Figure A.3: Biased vertices (ellipses) for rounding. Wavy edges are matched.

implies $3(i-2) \geq i$ shows

$$3ns + \sum_{i \geq 3} ic_i \leq 3\delta.$$

Using this with (A.7) shows $\leq 3\delta - ns$ vertices are not c-singletons. □

We can now give the game plan for the approximation algorithm. Fig.A.3 illustrates this, although some of the notation isn't defined until next section. The goal is to build a big lead. Call a vertex *biased* if, as in Lemma 25(i), its fractional edges have bias ≥ 1; otherwise the vertex is *unbiased*. Recall that $\mathcal{S}.$ is the set of all singletons. Define

$$B = \{ \text{ all biased singletons } \}, \quad C = \{ \text{ all c-singletons } \}.$$

The vertices of B can be rounded to build a lead. This doesn't help if B is small. But then $\mathcal{S}. - B$ is big, as is $C - B$ (Lemma 26). The vertices of $\mathcal{S}. - B$ have $d_F(v)$ even (Lemma 25(i)). We will make many of these degrees odd by rounding 1 heavy edge incident to v (Lemma 25(ii)). We then solve the residual LP, more precisely LPR to preserve the biased vertices of B. There are then several possibilities, all contributing to build the desired lead:

LR edges automatically increase the lead. Rounded vertices that become singletons in the new laminar family, and are not on LR edges, are biased ($d_F(v)$ is odd). Similarly vertices of B not on LR edges remain biased. These biased vertices all increase the lead. Finally rounded vertices that become nonsingletons shrink the size of the laminar family, i.e., they reduce the number of fractional edges. This also increases the lead.

To ensure the above shrinking is efficient we select the rounded edges as a big matching on $\mathcal{S}. - B$. The matching is big because of c-singletons, which guarantee more heavy edges (Lemma 25(ii)). It is useful to define subsets of $\mathcal{S}. - B$ based on fractional degrees $d_F(v)$. Recall that all vertices of $\mathcal{S}. - B$ have even fractional degree. Define

$$\begin{aligned} S_2 &= \{v : v \text{ an unbiased singleton, } d_F(v) = 2\}, \\ S^2 &= \{v : v \text{ an unbiased singleton, } d_F(v) \geq 2\}, \\ S^4 &= \{v : v \text{ an unbiased singleton, } d_F(v) \geq 4\}, \end{aligned}$$

and define C_2 and C^2 as analogous subsets of c-singletons. So for instance $\mathcal{S}. - B = S^2 = S_2 \cup S^4$, $C = C^2$. The matching will be guaranteed to be big as long as C_2 is small (C_2 vertices impede the matching because they have too few edges). So rounding the matching doesn't help if C_2 is big. This isn't a problem when the requirement function f (or the connectivity k) is even – Lemma 25(iii) shows $C_2 = \emptyset$. So the even connectivity algorithm is simpler. It is presented in the next section. The general f (odd k) algorithm uses a second routine for C_2 vertices, and is presented after that.

A.3.4 Rounding by Matching

This section presents the main approximation algorithm. It rounds a matching on even degree singletons. The algorithm applies to the SN problem with general requirement function f, so that problem is assumed throughout the discussion. A factor $1 + 1.89/k$ approximation for the even k-ECSS problem follows immediately from this section's algorithm.

The algorithm is given in Fig.A.4. It begins by solving $LP(\emptyset)$, as in iterated rounding (Fig.A.4 line 1). Throughout this section and the next we fix a parameter $\delta \geq 0$ so that the laminar family corresponding to $LP(\emptyset)$ contains exactly $2n-\delta$ sets. We also continue to use the notation introduced at the end of last section (Fig.A.3). So line 2 of the algorithm fixes the set of biased vertices that will be preserved by LPR (line 4).

Since we want to bias the even degree singletons, define the set of edges eligible for matching

1. solve $LP(\emptyset)$
2. $B =$ { all biased singletons }
3. round the edges of a maximum matching on EM to 1
4. solve $LPR(I)$
5. solve the residual problem by greedy rounding
   increasing the lead by half the total vertex bias

Figure A.4: Rounding by matching.

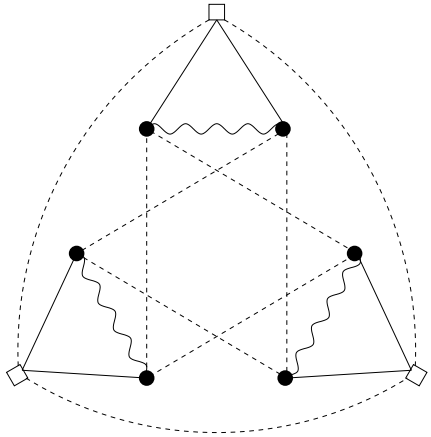


Figure A.5: Tight example for matching, for the even k-ECSS problem. Every vertex is in C_4. Solid and wavy edges weigh $1/2 + \epsilon$, dashed edges weigh $1/2 - \epsilon$.

to be

$$EM = \{e : e \text{ is a heavy edge joining 2 vertices of } S^2\}.$$

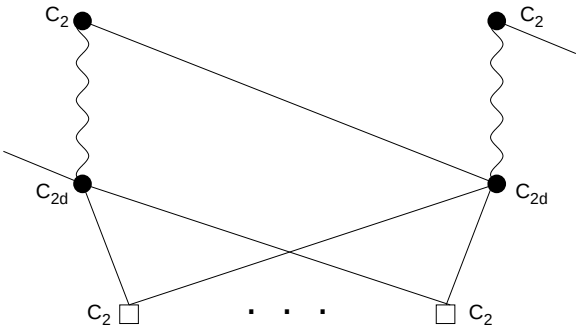
Line 3 finds a maximum cardinality matching on this edge set and rounds each matched edge to 1. Our main task is to analyze the size of this matching. We use some basic terminology for matchings that can be found in many texts, e.g. [112].

Lemma 27. *A maximum matching on EM has $\geq (n - 5\delta - 3|B| - |C_2|)/3$ edges.*

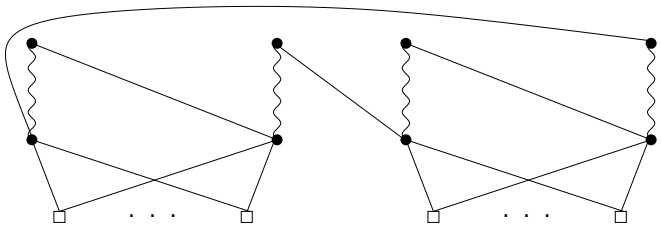
Remark The examples of Fig.A.5–A.6 illustrate the tightness of the lemma's bound, as well as the ideas of the lemma's proof. We will discuss these examples before proceeding to the proof. In these figures and all other figures that show matchings, wavy edges are matched and square vertices are unmatched.

The examples of the 2 figures are k-ECSS problems. By intent, we have not made any effort to ensure that the graphs of the examples correspond to a laminar family. However the examples do respect Corollary 5, which we restate as follows:

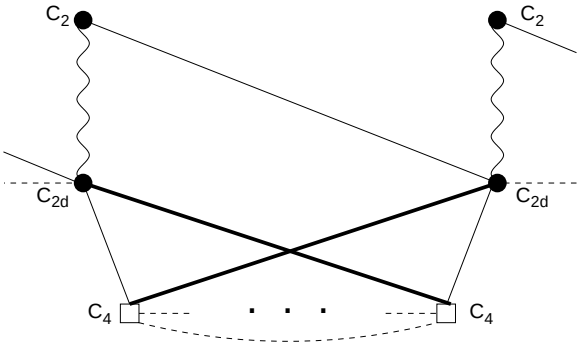
Fact *The edges incident to a c-singleton v can be partitioned into 2 sets, each set having total weight $k/2$.*



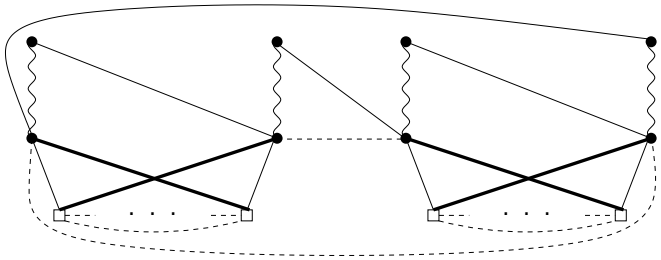
(a) 2-module. The bottom row has $2d-2$ C_2's.



(b) Subgraph of $r \geq 2$ 2-modules. Here $r = 2$.



(c) 4-module. The bottom row has $2d-3$ C_4's.



(d) Subgraph of two 4-modules.

Figure A.6: Tight example for matching, for the odd k-ECSS problem. The example consists of the subgraph of (b) and the subgraph of (d). Solid and wavy edges weigh 1/2, heavy edges weigh $1/2+\epsilon$, dashed edges weigh $1/2-\delta$.

When k is even the Fact means that the fractional edges incident to v can be partitioned into two sets of integral weight. This holds in Fig.A.5, since each vertex is on 2 edges of weight $1/2 + \epsilon$ and 2 edges of weight $1/2 - \epsilon$. When k is odd the Fact means that the fractional edges incident to v partition into 2 sets, each having weight congruent to $1/2$ modulo 1. Observe that this holds in Fig.A.6: In Fig.A.6(a) and (c), each C_2 vertex is on 2 edges of weight $1/2$, and each C_4 vertex is on 1 edge of weight $1/2$, 1 edge of weight $1/2 + \epsilon$, and 2 edges of weight $1/2 - \delta$ for $\delta = \epsilon/2$. In Fig.A.6(a) each C_{2d} vertex is on $2d$ edges of weight $1/2$. In Fig.A.6(c) each C_{2d} vertex is on 2 edges of weight $1/2$ from C_2 vertices. The Fact will be satisfied if the remaining $2d - 2$ edges weigh exactly $d - 1$. To achieve this, first note from Fig.A.6(d) that there are exactly 2 4-modules. Arrange the edges from C_4 vertices to C_{2d} vertices the same way in both of the 4-modules. As an example, in both 4-modules let the left C_{2d} be on $d - 1$ edges of weight $1/2 + \epsilon$ and $d - 2$ edges of weight $1/2$, and let the reverse hold for the right C_{2d}. Let one of the $1/2 - \delta$ edges join the 2 "left" C_{2d}'s. Define δ so that for both "left" C_{2d}'s, the edges from C_4 vertices and the $1/2 - \delta$ edge have total weight $d - 1$. Do the same for the 2 "right" C_{2d}'s. The value of δ on the right will be different than the left, but this is not a problem.

Now we will show that the lemma gives a tight bound on the size of the matching. Our examples have $\delta = |B| = 0$. The proof of Lemma 28 shows this is the case that determines the approximation ratio.

Consider first Fig.A.5, the example for even k. The heavy edges form 3 triangles. Note that the matching is maximum, since only 1 edge in each triangle can be matched. The example easily generalizes to an arbitrary number r of triangles of heavy edges. In terms of the notation of Lemma 27, the example has $n = 3r$, $\delta = |B| = |C_2| = 0$, and there are $r = n/3$ matched edges. So this example shows the bound of Lemma 27 is tight for the even k-ECSS problem.

Next consider Fig.A.6, the example for odd k. The matching is maximum since each C_{2d} vertex is on a unique matched edge, and those vertices form a vertex cover for the heavy edges.

Assume $d \geq 5$ and $r = d - 3$. The example has these parameters:

$$\begin{aligned} n &= r(4 + (2d - 2)) + 2(4 + (2d - 3)) = 2dr + 2r + 4d + 2 = 2dr + 6d - 4 \\ m &= r(2(2d)) + 2(2(2d) - 1 + (2d - 3)) = 4dr + 12d - 8 = 2n \\ |C_2| &= r(2 + (2d - 2)) + 4 = 2dr + 4 \end{aligned}$$

Since $m = 2n$ we have $\delta = 0$, and also $|B| = 0$. Hence the bound of the lemma is a matching of $\geq (n - |C_2|)/3 = 2d - 8/3$ edges. The matching has $2r + 4 = 2d - 2$ edges. We can make d arbitrarily large, so the lemma's bound is tight.

Proof of Lemma 27: Let M be the set of matched vertices (see Fig.A.3). We claim

$$|S^2 - M| + |C^2 - M| \leq (4n - 2\delta) - 2|S_2| - 4|S^4| + |M| + |(S^2 - C^2) \cap M|. \tag{A.8}$$

We first show the claim implies the lemma, by simple algebra: Substituting $|S^2 - M| = |S^2| - |M|$ and $|C^2 - M| = |C^2| - |C^2 \cap M|$, and rearranging gives

$$|S^2| + |C^2| + 2|S_2| + 4|S^4| \leq 4n - 2\delta + 2|M| + |C^2 \cap M| + |(S^2 - C^2) \cap M| = 4n - 2\delta + 3|M|. \tag{A.9}$$

The left-hand side is at least

$$3|S^2| + 3|C^2| - 2|C_2|.$$

We will rewrite this expression using two identities:

$$|C^2| \geq n - 3\delta - |B|, \quad |S^2| \geq n - \delta - |B|.$$

The first of these follows from Lemma 26 which gives $|C| = |C^2| + |B \cap C| \geq n - 3\delta$. The second follows since $n - (|S^2| + |B|)$ equals the number of nonsingletons, which is obviously $\leq \delta$. So we get that the left-hand side is at least $6n - 12\delta - 6|B| - 2|C_2|$. Recalling the right-hand side of (A.9) we get

$$2n - 10\delta - 6|B| - 2|C_2| \leq 3|M|.$$

Remembering that the number of matched edges is $|M|/2$, we get the bound of the lemma.

We turn to proving (A.8). Recall that every unmatched vertex of S^2 (C^2) is on ≥ 1 (≥ 2) heavy edges respectively (Lemma 25(ii)). So the left-hand side of (A.8) lower bounds the number of heavy edges incident to unmatched vertices of S^2. We will show that the right-hand side upper bounds this quantity, by using a system of credits. Give each vertex v a number of credits equal to

$$\begin{cases} d_F(v) & \text{if } v \notin S^2 \\ d_F(v) - 3 \geq 1 & \text{if } v \in C^4 \cap M \\ d_F(v) - 2 \geq 2 & \text{if } v \in (S^4 - C^4) \cap M \\ d_F(v) - 1 = 1 & \text{if } v \in C_2 \cap M \\ d_F(v) = 2 & \text{if } v \in (S_2 - C_2) \cap M. \end{cases}$$

Notice that unmatched vertices of S^2 get no credits. We begin by showing that the total number of credits is at most the right-hand side of (A.8): The sum of all degrees $d_F(v)$ is equal to $2|F| = 2|\mathcal{L}.| = 4n - 2\delta$. The number of credits equals this sum of all degrees decreased by these quantities:

$d_F(v)$ for $v \in S^2 - M$, 3 for $v \in C^4 \cap M$, 2 for $v \in (S^4 - C^4) \cap M$, and 1 for $v \in C_2 \cap M$.

The total decrease is

$$\geq (4|S^4| - |C^4 \cap M| - 2|(S^4 - C^4) \cap M|) + (2|S_2| - |C_2 \cap M| - 2|(S_2 - C_2) \cap M|)$$
$$= 4|S^4| + 2|S_2| - |M| - |(S^2 - C^2) \cap M|.$$

This implies the number of credits is upper bounded by the right-hand side of (A.8).

The rest of the argument shows that the number of credits upper bounds the left-hand side of (A.8), by paying 1 credit for each heavy edge incident to an unmatched vertex of S^2. We work with a graph H constructed from G in 3 steps:

1. Discard all edges that are not heavy.

2. If the current graph contains an alternating path P that joins vertices $v, w \in S^2 - M$, use the credits of vertices on P to pay for the first and last edge of P. Then delete the vertices of P from H. Repeat this step until no such P exists.

3. Split each vertex $v \in S^2 - M$ that remains in the graph into degree 1 vertices, collectively

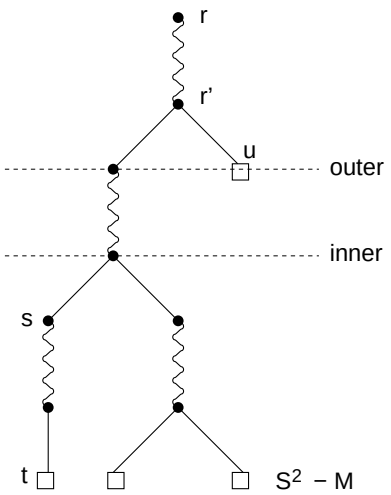


Figure A.7: Alternating tree. Vertex labels illustrate the last part of the proof of Lemma 27.

indicent to the neighbors of v. We continue to refer to these new vertices as vertices of $S^2 - M$.

As an example, in Fig.A.5 steps 1–2 pay for every heavy unmatched edge. In general in step 2, the alternating path P contains at least 1 matched edge, since the matching is maximum. (In fact we must have $v = w$ for the same reason.) Since each matched vertex has a credit, we can pay for the 2 unmatched ends of P as required. (As a minor comment note that if v gets deleted in this step, we will only pay for 2 heavy edges incident to v. This suffices to establish (A.8). With slightly more care we could actually pay for every heavy edge incident to v.)

Step 2 does not delete any edge incident to a vertex of $S^2 - M - \{v, w\}$ (such an edge would give an augmenting path). So to complete the argument it suffices to pay for the edges of H that are incident to vertices of $S^2 - M$.

The argument is based on a notion of alternating tree that we now define, illustrated in Fig.A.7. (As can be seen in the figure, our definition differs slightly from the usual one, which is oriented towards searching for an augmenting path.) An *alternating tree* in graph H is a tree whose vertices are labelled "inner" and "outer", such that

i) in any path from a leaf to the root, the vertices alternate between inner and outer;

ii) each edge from an inner vertex to its (outer) parent is matched;

iii) each leaf is in $S^2 - M$ and is outer;

iv) the root is either outer, or an inner vertex not in S^2.

An alternating tree is *deficient* if the credits in its vertices cannot pay for all of its leaves.

Claim 1 *Let T be a deficient tree, with root r.*

i) T can pay for every leaf but 1; furthermore r is outer.

ii) Every matched inner vertex v has all its incident fractional edges in T; furthermore $v \in C^4$.

iii) If r is matched then $r \in C^2$.

Proof: i) Let T have ℓ leaves. In any tree $\ell - 1$ is equal to the sum, over all interior nodes v, of $c(v) - 1$, where $c(v)$ is the number of children of v. In T we need only sum over the inner nodes v. If v is matched then $c(v) \leq d_F(v) - 1$. So v contributes $\leq d_F(v) - 2$ to the sum, and together with its outer parent v has $\geq (d_F(v) - 3) + 1 = d_F(v) - 2$ credits. Hence if r is outer and matched T has $\geq \ell - 1$ credits, as claimed. This also holds if r is outer and unmatched, i.e., T consists of just a leaf. In the remaining case r is inner, $c(r) \leq d_F(r)$ and r has $d_F(r)$ credits. Thus T has $\geq \ell$ credits, as claimed.

ii – iii) For a deficient tree, every inequality in the above argument must hold with equality. To show the second part of (ii) note that v has $> d_F(v) - 3$ credits unless $v \in C^4$. To show (iii) note that r has ≥ 2 credits if $r \in S^2 - C^2$. $\diamondsuit$

We will grow a forest F in the graph H. F will consist of nondeficient alternating trees that collectively contain all vertices of $S^2 - M$. Nondeficiency implies such a forest completes the proof. To grow F start by making every vertex of $S^2 - M$ a leaf, the root of its own alternating tree. Every such tree is deficient. Then repeat the following step as long as some tree remains deficient.

Grow Step. Choose a deficient tree, with root r. Add (r, s), an edge of H incident to r, to F. If s is matched then add its matched edge to F unless it is already in F.

As an example, in Fig.A.6 the final forest consists of $2r + 4$ trees, each tree consisting of a

matched edge and the $2d-2$ or $2d-3$ edges leading to the unmatched neighbors of the C_{2d} vertex. Each of the $2r$ trees in Fig.A.6(b) pays for its unmatched edges with no credits left over. Each of the 4 trees in Fig.A.6(d) has 1 extra credit.

We must show that the Grow Step can always be executed, and it results in a valid alternating forest. This will complete the proof, since it is clear that the procedure eventually halts with no deficient tree.

First note that the desired heavy unmatched edge (r, s) always exists. If r is matched then $r \in C^2$ (Claim 1(iii)) and so Lemma 25(ii) guarantees existence of the edge. Second, edge (r, s) always belongs to H, i.e., s was not deleted in step 2, since a deleted s would give an augmenting path for the matching.

It remains only to show that the Grow Step maintains F as a forest. This is clear if s does not already belong to a tree of F, so assume the opposite. Let T be the tree of F containing r at the start of the Grow Step.

If s is inner then $s \notin T$, by Claim 1(ii). So adding (r, s) joins 2 trees of F and results in a valid forest.

We complete the proof by showing that s is never outer. Suppose on the contrary that s is outer. We will exhibit an alternating path P in H between 2 vertices of $S^2 - M$. But this violates step 2 of the construction of H.

Vertex s may belong to T or a different tree. But P is constructed the same way in both cases. The case $s \in T$ is illustrated in Fig.A.7. Assume for the moment that T is not a singleton tree, so its root r is matched, say to vertex r'. r' is inner and it has ≥ 2 children. This follows from Claim 1(ii) which actually implies it has $d_F(r') - 1 \geq 3$ children. Let t be a leaf descending from s (possibly $t = s$) and let u be a leaf descending from r' but not s. Form P by combining the path in T from u to r', edges (r', r) and (r, s), and the tree path from s to t. Clearly P is the desired alternating path. If T is a singleton tree the construction is simpler – P consists of edge (r, s), and the tree path from s to t. □

This algorithm achieves a good lead when C_2 is small.

Lemma 28. *The algorithm of Fig.A.4 halts with a lead* $\geq (n - |C_2|)/9$.

Proof. We continue to use M as the set of all matched vertices. Let x be the extreme point solution to $LPR(I)$ in Fig.A.4 line 4. Use x to define F, LR, etc. Note that x also defines a laminar family $\mathcal{L}.$ by Lemma 23(iii).

We introduce the remaining notation in Fig.A.3. Consider a vertex $v \in M$. After the rounding of line 3, v is on an odd number of fractional edges (a vertex of S^2 has even d_F). So $d_F(v)$ is odd unless v is on one or more edges of LR. Similarly Lemma 23(ii) guarantees that a vertex $v \in B$ has bias ≥ 1 after line 4 unless v is on an edge of LR. With this in mind define

$$\begin{aligned} MLR &= \{v : v \in M,\ v \text{ is on an edge of } LR\ \}, \\ BLR &= \{v : v \in B,\ v \text{ is on an edge of } LR\ \}. \end{aligned}$$

Partition the rest of M into 2 subsets:

$$MS = (M - MLR) \cap \mathcal{S}., \quad MNS = M - MLR - \mathcal{S}..$$

Let λ_I be the lead immediately after line 4 and let λ_F be the final lead. Recall that each step of the algorithm preserves the lead (e.g., Lemma 23(i)).

We first show

$$\lambda_I \geq \max\{\delta + |LR|,\ |M|/2 - |MLR| - |MS|\}. \tag{A.10}$$

The first estimate in the max is simply the initial lead plus the lead from fractional edges that $LPR(I)$ makes integral. For the second estimate note that after solving $LPR(I)$ we have $|F| \leq 2n - |MNS|$. Since line 3 rounds a matching of $|M|/2$ heavy edges, $2\Delta \leq |M|/2$. This gives lead $\geq |MNS| - |M|/2$. (The values of δ and $|LR|$ do not improve this estimate!) Rewriting $|MNS|$ in terms of $|M|$ from the partition equation $|M| = |MLR| + |MS| + |MNS|$ gives the second estimate in the max.

Since the maximum is at least the average, (A.10) gives

$$\lambda_I \geq |M|/4 + (\delta + |LR| - |MLR| - |MS|)/2.$$

After line 4, $|B - BLR|$ vertices from line 2 still have bias ≥ 1. In addition $|MS|$ singletons with bias ≥ 1 have been created (Lemma 25(i)). So adding in half the total bias, Lemmas 21–22 show line 5 gives a final lead

$$\lambda_F \geq |M|/4 + (\delta + |LR| - |MLR| + |B - BLR|)/2.$$

Since each vertex of $MLR \cup BLR$ is on an edge of LR, $2|LR| \geq |MLR| + |BLR|$. So the previous estimate becomes

$$\lambda_F \geq |M|/4 + (\delta + |B| - |LR|)/2.$$

Recalling the first term of (A.10),

$$\lambda_F \geq \delta + |LR|,$$

the two inequalities combine to give

$$3\lambda_F \geq |M|/2 + 2\delta + |B|.$$

The lower bound for $|M|/2$ of Lemma 27 shows $3\lambda_F \geq (n + \delta - |C_2|)/3$. This gives the bound of the lemma. □

The lead is $\geq n/9$ for the even-SN problem, since $C^2 = \emptyset$ (Lemma 25(iii)). So the definition of lead gives the following.

Corollary 6. *For any even k the algorithm of Fig.A.4 approximates the smallest k-ECSS of an undirected multigraph to within a factor $1 + 17/9k < 1 + 1.89/k$. The same bound holds for any even-SN problem with average vertex demand k.*

A.3.5 Rounding by Covering

This section shows that the algorithm of Fig.A.8 gives a good approximation when C_2 is large. After analyzing the algorithm we show how it combines with the algorithm of last section to always achieve the desired approximation ratio. Throughout the discussion we assume the problem is the SN problem with arbitrary f.

1. solve $LP(\emptyset)$
2. round a minimal set of heavy edges covering T_2 to 1
3. solve the residual problem by iterated rounding

Figure A.8: Rounding by covering.

The algorithm does not use properties specific to c-singletons so we state it in the more general way. The algorithm starts by finding an extreme point solution x to $LP(\emptyset)$. Define

$$T_2 = \{v : d_x(v) = f(v),\ d_F(v) = 2\},$$

the "tight" vertices of degree 2. Clearly $C_2 \subseteq T_2$. Also every vertex of T_2 is on a heavy edge. Hence the cover of line 2 exists. We will show the lead after line 2 is $|T_2|/2$. Line 3 preserves this lead.

Let R be the cover that gets rounded. Let x' be the extreme point found right after rounding R. Use x' to define the laminar family $\mathcal{L}.'$ and all tree terminology, as well as F' and Δ'.

Lemma 29. *If S is an $\mathcal{L}'$.-set contained in T_2 then R contains an edge with both ends in S.*

Proof: Let w be the fractional weight function of x. Then $d_x(S) \geq f(S)$ and $d_w(S) \leq |S|$ (since each $v \in S$ has $d_w(v) = 1$). These two inequalities imply that $\geq f(S) - |S|$ integral edges of x leave S. $S \in \mathcal{L}.'$ implies that after rounding, $< f(S)$ integral edges leave S. Thus $< |S|$ edges leaving S are rounded in line 2. So some vertex of S has a rounded edge that does not leave S, □

Lemma 30. *The algorithm of Fig.A.8 halts with a lead $\geq |T_2|/2$.*

Proof. We show the desired lead is achieved right after x' is found. Recalling that all tree terminology refers to $T(\mathcal{L}.')$, partition T_2 into 5 sets:

$$\begin{aligned}
P_2T &= \{v : v \in T_2,\ \text{the parent of } v \text{ has 2 children, both belonging to } T_2\};\\
P_2\overline{T} &= \{v : v \in T_2,\ \text{the parent of } v \text{ has 2 children, with only } v \text{ belonging to } T_2\};\\
P_3T &= \{v : v \in T_2,\ \text{the parent of } v \text{ has 3 children, all belonging to } T_2\};\\
P_3\overline{T} &= \{v : v \in T_2,\ \text{the parent of } v \text{ has 3 children, not all belonging to } T_2\};\\
P^4 &= \{v : v \in T_2,\ \text{the parent of } v \text{ has} \geq 4 \text{ children}\}.
\end{aligned}$$

We will prove 3 properties:

i) $\mathcal{L}.'$ has $\geq |T_2| + |P_2\overline{T}|$ nonsingleton vertices.

ii) $\mathcal{L}.'$ has $\leq n - (|P_3\overline{T} \cup P^4|/2 + |P_3T|/3)$ nonsingleton sets.

iii) R contains a matching M of $\geq |P_2T|/2 + |P_3T|/3$ edges with both ends in T_2.

We first verify that these properties imply the desired lead. (i) and (ii) show that the number of fractional edges in x' is at most

$$|\mathcal{L}.'| \leq 2n - (|T_2| + |P_2\overline{T}|) - (|P_3\overline{T} \cup P^4|/2 + |P_3T|/3).$$

Since R is a minimal cover of T_2, it has $\leq |M| + (|T_2| - 2|M|) = |T_2| - |M|$ edges. So (iii) shows

$$\Delta' \leq |R|/2 \leq \big(|T_2| - (|P_2T|/2 + |P_3T|/3)\big)/2.$$

Combining the previous 2 inequalities shows the lead is

$$\begin{aligned} &\geq \quad (|T_2| + |P_2\overline{T}|) + (|P_3\overline{T} \cup P^4|/2 + |P_3T|/3) - \big(|T_2| - (|P_2T|/2 + |P_3T|/3)\big) \\ &= \quad |P_2\overline{T}| + |P_2T \cup P_3\overline{T} \cup P^4|/2 + 2|P_3T|/3 \ \geq \ |T_2|/2 \end{aligned}$$

as desired.

We establish properties (i)–(iii) by showing that each of the 5 subsets of T_2 makes the appropriate contribution to the 3 quantities. The matching M will be a constructed as a collection of edges of R that join 2 vertices with the same parent in $T(\mathcal{L}'.)$; M will contain ≤ 1 edge per parent, which guarantees it is a matching.

First note that every $v \in T_2$ is a nonsingleton of $\mathcal{L}.'$, since $d_{F'}(v) \leq 1$. This gives the first term in (i).

A vertex $v \in P_2\overline{T}$ has a unique sibling s. s is not an $\mathcal{L}.'$-set, since $d_{F'}(v) \leq 1$. Hence s is a vertex that is a nonsingleton in $\mathcal{L}.'$. This gives the second term of (i), since the definition of $P_2\overline{T}$ shows $s \notin T_2$.

Next we show (iii). The vertices of P_2T come in pairs. Lemma 29 shows each pair is joined by an edge of R. Placing this edge in the matching M gives the first term of (iii). Similarly the vertices of P_3T come in triples and Lemma 29 shows each triple induces an edge of R. This gives the second term of (iii).

To show (ii), note that for any $c \geq 3$, an $\mathcal{L}'.$-set with c children reduces the total number of nonsingleton sets in $\mathcal{L}'.$ by $c - 2$. So the parent of a vertex in P_3T reduces the number of

nonsingleton sets by 1 and it has 3 children in T_2. The parent of a vertex in $P_3\overline{T}$ reduces the number of nonsingleton sets by 1 and has ≤ 2 children in T_2. Finally ccnsider the parent of a vertex in P^4; let it have c children. If t of those children belong to T_2 we can say that each of them reduces the number of nonsingleton sets by $(c-2)/t \geq (c-2)/c \geq (4-2)/4 = 1/2$. This gives (ii). □

To summarize, the overall algorithm starts by solving $LP(\emptyset)$. If $|C_2| \geq 2n/11$ the algorithm of Fig.A.8 is used to complete the solution; in the opposite case Fig.A.4 is used. In both cases a lead $\geq n/11$ is achieved.

Theorem 3.8. *For any $k \geq 1$ the smallest k-ECSS of an undirected multigraph can be approximated to within a factor $1+21/11k < 1+1.91/k$. The same bound holds for any SN problem with average demand k.*

A.4 Simple Graph Algorithm

This section presents an implementation of iterated rounding that improves our approximation ratio for k-ECSS on simple graphs with $k \geq 13$. The entries of Table A.1 for $k = 100, 200, 400$ are for this algorithm.

The main idea was introduced implicitly in Section A.2.2 and is formalized by the following definition: A vertex v is **satisfied** when $d_I(v) \geq k$, where (as in Section A.3.2) I denotes the total weight of edges that have fixed integral values and are incident to v. Once satisfied, a vertex remains satisifed for the rest of the algorithm, and so it must always be a nonsingleton. In contrast, a vertex can in general oscillate back and forth between singleton and nonsingleton in various iterations.

We will analyze our algorithm by charging increases in the LP objective function to vertices that get satisfied. We will use this sufficient condition: A singleton becomes satisfied if all but at most 1 of its fractional edges gets rounded.

Since our analysis tracks Δ, the increase in the objective function, we need this modification of the results of Section A.3.2: If greedy rounding is executed, starting with a set of fractional edges

F_0 having total vertex bias B, then Δ increases by at most

$$|F_0|/2 - B/4.$$

In proof, define the lead to be the quantity

$$\lambda = |F_0| - |F| - 2\Delta.$$

We take Δ to be 0 when greedy rounding begins, so the lead starts at 0. Lemmas 21–22 clearly hold for the new definition of lead. They show greedy rounding increases the lead by at least the total edge bias, which is $\geq B/2$. At the end of the algorithm $F = \emptyset$, so $2\Delta = |F_0| - \lambda \leq |F_0| - B/2$, as desired.

We will present 2 versions of our algorithm. The first version almost achieves the desired approximation ratio, and its analysis is simpler. The second version adds one additional rule, and follows a similar analysis to get our best approximation ratio.

A.4.1 The Basic Algorithm

To state the algorithms first recall the notation of Section A.2.2. As before we partition a laminar family $\mathcal{L}.$ into the singleton sets $\mathcal{S}.$ and nonsingleton sets $\mathcal{N}.$. We further partition $\mathcal{S}.$ into 3 sets, the set S_2 defined as before and

$$S_3 = \{v : v \in \mathcal{S}.,\ d_F(v) = 3\}, \quad S^4 = \{v : v \in \mathcal{S}.,\ d_F(v) \geq 4\}.$$

A **pairing edge** is a fractional edge joining 2 vertices of S_2. A *touching edge* is a fractional edge incident to exactly 1 vertex of S_2.

The basic algorithm is presented in Fig.A.9. It is divided into 3 stages. Stage I corresponds to the beginning of the algorithm of Section A.2.2. Observe that in Stage I, the increase in the objective function can be accounted for by charging 1/4 to each vertex that gets satisfied. Specifically each pairing edge that gets rounded satisfies 2 vertices and increases the objective by $\leq 1/2$, so each vertex gets charged 1/4.

0. *Stage I:* solve $LP(\emptyset)$
1. repeat until no edge gets rounded:
2. round a maximal matching of heavy pairing edges
3. solve $LP(I)$
4. *Stage II:* repeat until no edge gets rounded:
5. round every touching edge of weight $\geq 2/3$,
   and a maximal matching of pairing edges of weight $> 1/3$
6. solve $LP(I)$
7. *Stage III:* solve the residual problem by greedy rounding

Figure A.9: Basic algorithm for simple graphs.

In Stage II we bound the increase in objective function by charging 1/3 to each vertex that gets satisfied. Specifically, an S_2 vertex on a touching edge gets charged the increase in Δ, which is $\leq 1/3$. The two S_2 vertices on a pairing edge that gets rounded share the increase in Δ, so the charge to each is $< 1/3$. Finally note that a vertex gets charged only once – an S_2 vertex is on ≤ 1 touching edge, and on ≤ 1 pairing edge that gets rounded, and it cannot be on both.

In Stage III we use the vertex bias to bound the increase in Δ, as indicated in the introduction to this section.

Lemma 31. *For any $k \geq 7$, the algorithm of Fig.A.9 approximates the smallest k-ECSS of a simple graph to within a factor $1 + 1/2k + 11\nu(k)/4k$.*

Proof. Consider the last iteration of Stage I. No edge gets rounded in this iteration. This iteration was analyzed in the proof of Theorem 2.7, since it corresponds to the first iteration where there are no good edges in the algorithm of Section A.2.2. So our proof begins by redoing the analysis of Section A.2.2. Towards that end, let $\mathcal{L}. = \mathcal{S}. \cup \mathcal{N}.$ be the laminar family in the last iteration of Stage I. Let all terms like S^3, F etc. refer to $\mathcal{L}.$. We shall also count nonsingleton vertices, so define for every $i \geq 0$,

$$ns_i = |\{v : v \notin \mathcal{S}.,\ d_F(v) = i\}|, \quad ns^i = |\{v : v \notin \mathcal{S}.,\ d_F(v) \geq i\}|.$$

Analogous to (A.5) we have $2|S_2| + 3|S_3| + 4|S^4| + \sum_{i \geq 0} ins_i \leq 2(|\mathcal{S}.| + |\mathcal{N}.|)$. Thus

$$|S_3| + 2|S^4| + \sum_{i \geq 0} ins_i \leq 2|\mathcal{N}.|. \tag{A.11}$$

To state a refined version of (A.6) partition S_2 into 2 sets,

$$S_{21} = \{v : v \in S_2,\ d_F(v, V - S_2) = 1\}, \quad S_{22} = \{v : v \in S_2,\ d_F(v, V - S_2) = 2\}.$$

(Fig.A.10 illustrates these sets, as well as other sets that will be used in the proof of the refined algorithm.) This is a partition since Section A.4 shows that at the end of Stage I every vertex of S_2 is on a fractional edge leading to a vertex not in S_2. Now (A.6) translates to

$$|S_{21}| + 2|S_{22}| \leq 2(|S^3| + |\mathcal{N}.|). \tag{A.12}$$

Next we analyze the end of Stage II. Let $\mathcal{L}.' = \mathcal{T}. \cup \mathcal{N}.'$ be the laminar family in the last iteration of Stage II. Define sets of singletons T_2, T_3, T^3, T^4 as the subsets of $\mathcal{T}.$ analogous to S_2, S_3 etc.

Claim 1 *No pairing edge exists at the end of Stage II.*

Proof: Any such pairing edge weighs $\leq 1/3$ (it cannot be rounded in Stage II). If $v \in T_2$ is on a pairing edge, consider the other fractional edge incident to v, say (v, x). Clearly (v, x) weighs $\geq 2/3$. Furthermore we have shown (v, x) is not pairing, so it is a touching edge. But that means it can be rounded. ♢

The claim shows that any $v \in T_2$ is on 2 fractional edges leading to $V - T_2$. So analogous to (A.12) we have $2|T_2| \leq 2(|T^3| + |\mathcal{N}.'|)$, i.e.,

$$|T_2| \leq |T^3| + |\mathcal{N}.'|. \tag{A.13}$$

Claim 2 *Any vertex $v \in S_{21}$ is on an edge that gets rounded in the first iteration of Stage II.*

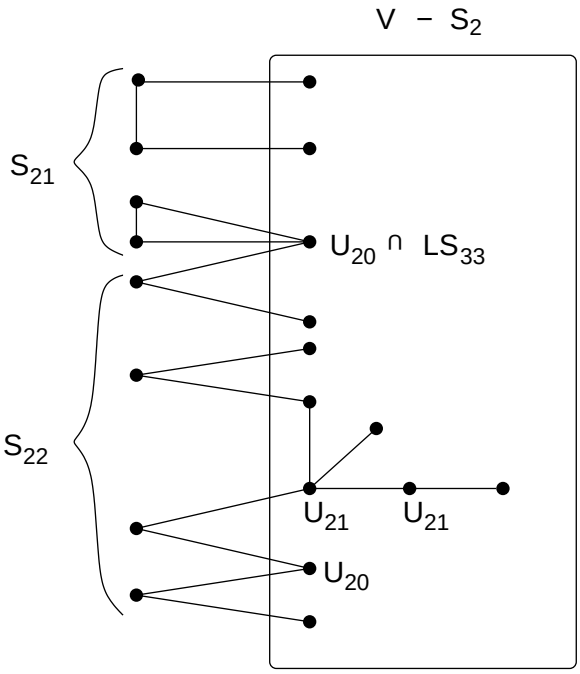


Figure A.10: Sets at the end of Stage I, in the proofs of Section A.4.

Proof: Any pairing edge at the end of Stage I weighs $< 1/2$ (it cannot be rounded). Hence no two pairing edges have a common vertex. Take any $v \in S_{21}$, and let it be on the pairing edge (v, w). We have shown the other fractional edge incident to v is a touching edge. The same holds for w (which is also in S_{21}). Now either the pairing edge (v, w) weighs $> 1/3$ and it gets rounded in the first iteration of Stage I, or both touching edges weigh $\geq 2/3$ and they get rounded in the first iteration. $\diamondsuit$

Since every singleton $v \in \mathcal{T}.$ is on ≥ 2 fractional edges at the end of Stage II, the same holds at the end of Stage I. Furthermore Claim 2 shows $v \notin S_{21}$. So

$$|\mathcal{T}.| \leq |S_{22}| + |S^3| + ns^2. \tag{A.14}$$

And since every vertex of T^4 is on ≥ 4 fractional edges at the end of Stage II, the same holds at the end of Stage I, so

$$|T^4| \leq |S^4| + ns^4. \tag{A.15}$$

We turn to the objective function. Let Δ denote the total increase due to iterated rounding. We claim

$$\Delta \leq \frac{n - |\mathcal{S}.| - ns^2}{4} + \frac{|\mathcal{S}.| + ns^2 - |\mathcal{T}.|}{3} + \frac{|\mathcal{L}.'|}{2} - \frac{T_3}{4}. \tag{A.16}$$

Proof of (A.16): When Stage I ends exactly $|\mathcal{S}.| + ns^2$ vertices are on ≥ 2 fractional edges. Recall that we account for the increase in Δ by charging $1/4$ to each satisfied vertex, and such a vertex is on ≤ 1 fractional edge after its round. So $\leq n - (|\mathcal{S}.| + ns^2)$ vertices get charged in Stage I. Thus the first right-hand side term of (A.16) gives the total increase in Δ during Stage I.

Next we show that the second right-hand term accounts for the rounds of Stage II. Stage II charges $1/3$ to each satisfied vertex. So it suffices to show that $\leq |\mathcal{S}.| + ns^2 - |\mathcal{T}.|$ vertices get charged. As already noted when Stage I ends, exactly $|\mathcal{S}.| + ns^2$ vertices are on ≥ 2 fractional edges. Every vertex that is a singleton at some point during Stage II is included in this count. So each charged vertex is in this count, as is each vertex of $\mathcal{T}.$. Each charged vertex is on ≤ 1 fractional edge after its round, and so is not in $\mathcal{T}.$. Thus the number of charged vertices is $\leq |\mathcal{S}.| + ns^2 - |\mathcal{T}.|$ as desired.

When Stage II ends, each vertex of T_3 has bias 1. So the analysis of greedy rounding in the introduction to this section shows Stage III increases Δ by at most the last 2 terms of (A.16). $\diamondsuit$

Using $|\mathcal{L}.'| = |\mathcal{T}.| + |\mathcal{N}.'|$, (A.16) simplifies to

$$\Delta \leq \frac{n}{4} + \frac{|\mathcal{N}.'|}{2} + \frac{\overline{\Delta}.}{12} \qquad \text{for } \overline{\Delta}. = |\mathcal{S}.| + 2|\mathcal{T}.| - 3|T_3| + ns^2. \tag{A.17}$$

To bound the Stage II terms, i.e., $2|\mathcal{T}.| - 3|T_3| = 2T_2 - T_3 + 2T^4$, add together 3/2 times inequality (A.13), 1/2 times inequality (A.14), and 3 times inequality (A.15), to get

$$2|T_2| - |T_3| + 2|T^4| \leq \frac{1}{2}(3|\mathcal{N}.'| + |S_{22}| + |S_3| + 7|S^4| + ns^2 + 6ns^4).$$

Thus

$$\overline{\Delta}. \leq \frac{1}{2}(3|\mathcal{N}.'| + 2|S_{21}| + 3|S_{22}| + 3|S_3| + 9|S^4| + 3ns^2 + 6ns^4).$$

To bound the "S" terms add 7 times inequality (A.11) and 2 times inequality (A.12) to get

$$2|S_{21}| + 4|S_{22}| + 3|S_3| + 10|S^4| \leq 18|\mathcal{N}.| - 7\sum_{i\geq 0} ins_i.$$

Since $7\sum_{i\geq 0} ins_i \geq 14ns^2 \geq 3ns^2 + 6ns^4$, the last two inequalities combine to give $\overline{\Delta}. \leq (3|\mathcal{N}.'| + 18|\mathcal{N}.|)/2$. Recalling that $|\mathcal{N}.|, |\mathcal{N}.'| \leq \nu(k)n$ (Corollary 3), (A.17) gives

$$\Delta \leq \frac{n}{4} + \frac{33}{24}\nu(k)n = \frac{n}{4} + \frac{11}{8}\nu(k)n.$$

The theorem follows, since the smallest k-ECSS has $\geq kn/2$ edges. $\square$

A.4.2 The Refined Algorithm

To improve this algorithm we round additional edges in Stage I. For any extreme point solution to $LP(I)$ define

$$S_{33} = \{v : v \in S_3 \text{ and each of } v\text{'s 3 neighbors on fractional edges belongs to } S_2\}.$$

Partition S_{33} into two sets, the "light" vertices of weight 1 and the "heavy" vertices of weight 2:

$$LS. = \{v : v \in S_{33},\ d_w(v) = 1\}, \quad HS. = \{v : v \in S_{33},\ d_w(v) = 2\}.$$

0. *Refined Stage I:* solve $LP(\emptyset)$
1. repeat until no edge gets rounded:
2. round a heavy pairing edge, or round the 3 fractional edges incident to a vertex of $HS.$
3. solve $LP(I)$

Figure A.11: The refined algorithm uses this Stage I plus Stages II–III of Fig.A.9.

Here w is the fractional weight function of the extreme solution. Stage I for the new algorithm is given in Fig.A.11; Stages II–III are unchanged.

Note that the refined algorithm maintains the charging scheme of Stage I: When the edges of an $HS.$ vertex get rounded, all 4 vertices involved get satisfied and are left incident to ≤ 1 fractional edge. (The $HS.$ vertex itself will be on no fractional edge.) The objective function increases by exactly 1, since before the round the 3 edges have weight totalling to exactly 2. Thus the charge to each vertex is exactly 1/4.

Theorem 4.9. *For any $k \geq 7$, the algorithm of Fig.A.11 approximates the smallest k-ECSS of a simple graph to within a factor $1 + 1/2k + 5\nu(k)/2k$.*

Proof. Define $\mathcal{S}.$, $\mathcal{T}.$ and their associated sets exactly as before. Throughout this proof F denotes the set of fractional edges at the end of Stage I, and similarly w is defined at the end of Stage I. The analysis is based on analogs of inequalities (A.11)–(A.15) plus 2 additional inequalities. (A.11) is unchanged.

We use a tighter version of (A.12). Towards this end define $U_2 = T_2 - S_2$, i.e., U_2 consists of the vertices that become T_2 vertices during Stage II. Partition U_2 into 2 sets:

$$U_{20} = \{v : v \in U_2,\ d_F(v, V - S_2) = 0\}, \quad U_{21} = \{v : v \in U_2,\ d_F(v, V - S_2) \geq 1\}.$$

Fig.A.10 illustrates these sets. (The 2 vertices of U_2 with degree 2 were nonsingletons at the end of Stage I.) The new version of (A.12) is

$$|S_{21}| + 2|S_{22}| + 2|U_{21}| \leq 2(|S^3| + |\mathcal{N}.|). \tag{A.18}$$

Proof of (A.18): Consider the end of Stage I. The total degree (w.r.t. d_F) of all vertices of $V - S_2$ is $2(|\mathcal{S}.| + |\mathcal{N}.|) - 2|S_2| = 2(|S^3| + |\mathcal{N}.|)$, i.e., the right-hand side of (A.18). We show the left-hand side is a lower bound for this total degree.

Every vertex of S_{21} (S_{22}) is on 1 (2) fractional edges leading to a vertex not in S_2, respectively. In addition a vertex of U_{21} is on ≥ 1 fractional edge, neither end of which is in S_2. This gives the lower bound of the left-hand side. ♢

Inequality (A.13) holds unchanged. We rewrite (A.15) in a more precise form,

$$|T^4| \leq |S^4 \cap T^4| + ns^4. \tag{A.19}$$

We use a similar estimate for U_{20}:

$$|U_{20}| \leq |LS_{33}| + |S^4 \cap T_2| + ns^2. \tag{A.20}$$

Proof of (A.20): Consider the end of Stage I, and take any vertex of $v \in U_{20}$. The definition of U_2 shows $d_F(v) \geq 2$. So the last term of (A.20) accounts for the nonsingletons v. A singleton v must belong to $S_3 \cup S^4$ (by definition $v \notin S_2$). The next-to-last term of (A.20) accounts for the vertices $v \in S^4$.

The remaining possiblity is $v \in S_3$. The definition of U_{20} then implies $v \in S_{33}$. Furthermore we must have $v \in LS_{33}$, since $HS. = \emptyset$ at the end of Stage I. So the first right-hand side term of (A.20) accounts for the last possibility. ♢

We need one more inequality to get a good estimate on $|T_2|$:

$$|LS_{33}| + |S_2 \cap T_2| \leq |S_{22}|. \tag{A.21}$$

Proof of (A.21): Recall that $S_{21} \cap T_2 = \emptyset$. Hence $S_2 \cap T_2 = S_{22} \cap T_2$ and (A.21) is equivalent to $|LS_{33}| \leq |S_{22} - T_2|$. We show this by associating to each vertex $v \in LS_{33}$ a unique vertex $x \in S_{22} - T_2$.

As usual consider the end of Stage I. Since $d_w(v) = 1$ some fractional edge (v, x) weighs $\leq 1/3$. Let (x, y) be the other fractional edge incident to x; its weight is $\geq 2/3$ since $x \in S_2$. This implies vertex $y \notin S_2$, since (x, y) was not rounded in Stage I. Thus $x \in S_{22}$. Furthermore (x, y) is a touching edge, and it gets rounded in the first iteration of Stage II. Thus $x \notin T_2$. We associate v with x. Since x is on only 2 fractional edges, no other vertex of LS_{33} gets associated with x. $\diamondsuit$

To estimate $|T_2|$ first combine (A.20) and (A.21) to get $|U_{20}|+|S_2 \cap T_2| \leq |S_{22}|+|S^4 \cap T_2|+ns^2$. By definition T_2 is partitioned into sets $S_2 \cap T_2$, U_{20} and U_{21}. Hence

$$|T_2| \leq |S_{22}| + |S^4 \cap T_2| + ns^2 + |U_{21}|. \tag{A.22}$$

Turning to the objective function, inequalities (A.16)–(A.17) hold as before. As before we will derive an upper bound for $\overline{\Delta}$ .. Using (A.22) gives

$$\begin{aligned} \overline{\Delta}. &= |\mathcal{S}.| + 2|\mathcal{T}.| - 3|T_3| + ns^2 \\ &\leq (|T_2| - |T_3| + 2|T^4|) + |S_{21}| + 2|S_{22}| + |S^3| + |S^4 \cap T_2| + 2ns^2 + |U_{21}|. \end{aligned}$$

Add inequality (A.13), inequality (A.18), and 3 times inequality (A.19) to get

$$|S_{21}| + 2|S_{22}| + 2|U_{21}| + |T_2| - |T_3| + 2|T^4| \leq 2(|S^3| + |\mathcal{N}.|) + 3|S^4 \cap T^4| + 3ns^4 + |\mathcal{N}.'|.$$

Combining the last 2 inequalities and then using (A.11) shows $\overline{\Delta}.$ is at most

$$3|S^3| + 2|\mathcal{N}.| + 3|S^4| + 3ns^4 + |\mathcal{N}.'| + 2ns^2 \leq 8|\mathcal{N}.| - 3\sum_{i \geq 0} ins_i + 3ns^4 + |\mathcal{N}.'| + 2ns^2 \leq 8|\mathcal{N}.| + |\mathcal{N}.'|.$$

Since $|\mathcal{N}.|, |\mathcal{N}.'| \leq \nu(k)n$ (Corollary 3) we get $\overline{\Delta}. \leq 9\nu(k)n$. Thus

$$\Delta \leq \frac{n}{4} + (\frac{1}{2} + \frac{3}{4})\nu(k)n = \frac{n}{4} + \frac{5}{4}\nu(k)n.$$

The desired bound follows. $\square$

Appendix B

Full Results for Complex Survey

Here we give full results from our survey of MIPS complexes including all of the statistics that we measured. We first list those complexes which induced a connected subgraph in the Y2H data (connected complexes), then other complexes which do not induce a connected subgraph but have interactions between their proteins in the Y2H data (disconnected complexes). Finally, we include a list of complexes with no interactions between their proteins in our Y2H data.

B.1 Connected Complexes

For space reasons, statistics on complexes with MIPS ID numbers below 400 are in Tables B.1 and B.2. Those with IDs from 400 to 500 are in Tables B.3 and B.4. Complexes with IDs above 500 are in Tables B.5 and B.6.

Complex	Pro.	Type	n	m	ED	Max Deg.	Min Deg.	Ave. Deg.	Deg. Dev.	CC
90.10	3	Normal	3	2	0.67	2	1	1.33	0.47	0
110	4	Normal	4	4	0.67	3	1	2	0.71	0.6
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
133.20	6	Normal	6	5	0.33	5	1	1.67	1.49	0
140.30.30.30	3	Normal	3	3	1	2	2	2	0	1
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
260.20.10	4	Normal	4	3	0.5	2	1	1.5	0.5	0
260.70	3	Normal	3	2	0.67	2	1	1.33	0.47	0
270.20.10	3	Normal	3	3	1	2	2	2	0	1
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
270.20.20	3	Normal	3	2	0.67	2	1	1.33	0.47	0
270.20.30	9	Normal	9	26	0.72	7	2	5.78	1.62	0.82
		Haircut	9	26	0.72	7	2	5.78	1.62	0.82
		MHCS	7	20	0.95	6	5	5.71	0.45	0.95
270.20.40	4	Normal	4	6	1	3	3	3	0	1
		Haircut	4	6	1	3	3	3	0	1
		MHCS	4	6	1	3	3	3	0	1
300	3	Normal	3	3	1	2	2	2	0	1
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1

Table B.1: Statistics for MIPS complexes that induce a connected subgraph in the Y2H data. Statistics include MIPS complex ID, number of proteins in the complex, type of subgraph (either normal, haircut, or MHCS), number of proteins appearing in the Y2H data (n), and number of interactions in the Y2H data. Also gives the edge density (ED), degree statistics (max, min, average, and standard deviation), and clustering coefficient (CC).

Complex	Type	Ave. MCC	MCC Dev.	Max Bet.	Min. Bet.	Ave. Bet.	Bet. Dev	Edge Con.	Vert. Con.
90.10	Normal	1	0	1	0	0.33	0.47	1	1
110	Normal	1	0	2	0	0.5	0.87	1	1
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
133.20	Normal	1	0	10	0	1.67	3.73	1	1
140.30.30.30	Normal	1	0	0	0	0	0	2	2
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
260.20.10	Normal	0.5	0.5	2	0	1	1	1	1
260.70	Normal	1	0	1	0	0.33	0.47	1	1
270.20.10	Normal	1	0	0	0	0	0	2	2
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
270.20.20	Normal	1	0	1	0	0.33	0.47	1	1
270.20.30	Normal	0.9	0.19	3.77	0	1.22	1.17	2	2
	Haircut	0.9	0.19	3.77	0	1.22	1.17	2	2
	MHCS	1	0	0.2	0	0.14	0.09	5	5
270.20.40	Normal	1	0	0	0	0	0	3	3
	Haircut	1	0	0	0	0	0	3	3
	MHCS	1	0	0	0	0	0	3	3
300	Normal	1	0	0	0	0	0	2	2
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2

Table B.2: Statistics for MIPS complexes that induce a connected subgraph in the Y2H data. Statistics include MIPS complex ID, type of subgraph (either normal, haircut, or MHCS), mutual clustering coefficient statistics (average and standard deviation), betweenness statistics (max, min, average, and standard deviation), and edge and vertex connectivity.

Complex	Pro.	Type	n	m	ED	Max Deg.	Min Deg.	Ave. Deg.	Deg. Dev	CC
410.10	6	Normal	6	9	0.6	4	1	3	1	0.71
		Haircut	5	8	0.8	4	2	3.2	0.75	0.79
		MHCS	4	6	1	3	3	3	0	1
410.20	8	Normal	8	15	0.54	5	1	3.75	1.48	0.72
		Haircut	7	14	0.67	5	2	4	1.31	0.75
		MHCS	5	10	1	4	4	4	0	1
410.33	4	Normal	3	2	0.67	2	1	1.33	0.47	0
410.40.90	3	Normal	2	1	1	1	1	1	0	N/A
410.40.100	3	Normal	3	2	0.67	2	1	1.33	0.47	0
440.10.10	5	Normal	5	6	0.6	3	1	2.4	0.8	0.6
		Haircut	4	5	0.83	3	2	2.5	0.5	0.75
		MHCS	4	5	0.83	3	2	2.5	0.5	0.75
440.12.20	9	Normal	8	14	0.5	6	1	3.5	1.5	0.55
		Haircut	7	13	0.62	6	2	3.71	1.28	0.59
		MHCS	6	11	0.73	5	3	3.67	0.75	0.68
440.12.30	3	Normal	3	3	1	2	2	2	0	1
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
440.14.10	10	Normal	9	16	0.44	7	1	3.56	1.77	0.49
		Haircut	8	15	0.54	7	2	3.75	1.64	0.52
		MHCS	6	11	0.73	5	3	3.67	0.75	0.68
450	6	Normal	6	5	0.33	5	1	1.67	1.49	0
470.10	6	Normal	6	9	0.6	5	2	3	1.15	0.55
		Haircut	6	9	0.6	5	2	3	1.15	0.55
		MHCS	6	9	0.6	5	2	3	1.15	0.55
470.20	5	Normal	5	8	0.8	4	3	3.2	0.4	0.67
		Haircut	5	8	0.8	4	3	3.2	0.4	0.67
		MHCS	5	8	0.8	4	3	3.2	0.4	0.67
470.30.10	3	Normal	3	2	0.67	2	1	1.33	0.47	0
480.10.05	3	Normal	3	3	1	2	2	2	0	1
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1

Table B.3: Statistics for MIPS complexes that induce a connected subgraph in the Y2H data. Statistics include MIPS complex ID, number of proteins in the complex, type of subgraph (either normal, haircut, or MHCS), number of proteins appearing in the Y2H data (n), and number of interactions in the Y2H data. Also gives the edge density (ED), degree statistics (max, min, average, and standard deviation), and clustering coefficient (CC).

Complex	Type	Ave. MCC	MCC Dev.	Max Bet.	Min. Bet.	Ave. Bet.	Bet. Dev.	Edge Con.	Vert. Con.
410.10	Normal	0.74	0.35	4	0	1.33	1.49	1	1
	Haircut	1	0	1	0	0.4	0.49	2	2
	MHCS	1	0	0	0	0	0	3	3
410.20	Normal	0.7	0.37	6	0	2.38	2.1	1	1
	Haircut	0.9	0.22	2	0	1.14	0.99	2	2
	MHCS	1	0	0	0	0	0	4	4
410.33	Normal	1	0	1	0	0.33	0.47	1	1
410.40.90	Normal	N/A	N/A	0	0	0	0	1	1
410.40.100	Normal	1	0	1	0	0.33	0.47	1	1
440.10.10	Normal	0.78	0.34	3	0	1	1.1	1	1
	Haircut	1	0	0.5	0	0.25	0.25	2	2
	MHCS	1	0	0.5	0	0.25	0.25	2	2
440.12.20	Normal	0.75	0.33	6.33	0	2.13	2.47	1	1
	Haircut	0.86	0.21	4.17	0	1.14	1.44	2	2
	MHCS	0.83	0.21	1.67	0	0.67	0.54	3	3
440.12.30	Normal	1	0	0	0	0	0	2	2
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
440.14.10	Normal	0.75	0.34	10	0	2.67	3.48	1	1
	Haircut	0.85	0.22	7.67	0	1.63	2.47	2	2
	MHCS	0.83	0.21	1.67	0	0.67	0.54	3	3
450	Normal	1	0	10	0	1.67	3.73	1	1
470.10	Normal	0.9	0.2	4	0	1	1.44	2	2
	Haircut	0.9	0.2	4	0	1	1.44	2	2
	MHCS	0.9	0.2	4	0	1	1.44	2	2
470.20	Normal	0.8	0.24	0.67	0.33	0.4	0.13	3	3
	Haircut	0.8	0.24	0.67	0.33	0.4	0.13	3	3
	MHCS	0.8	0.24	0.67	0.33	0.4	0.13	3	3
470.30.10	Normal	1	0	1	0	0.33	0.47	1	1
480.10.05	Normal	1	0	0	0	0	0	2	2
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2

Table B.4: Statistics for MIPS complexes that induce a connected subgraph in the Y2H data. Statistics include MIPS complex ID, type of subgraph (either normal, haircut, or MHCS), mutual clustering coefficient statistics (average and standard deviation), betweenness statistics (max, min, average, and standard deviation), and edge and vertex connectivity.

Complex	Pro.	Type	n	m	ED	Max Deg.	Min Deg.	Ave. Deg.	Deg. Dev	CC
510.20	4	Normal	4	3	0.5	3	1	1.5	0.87	0
510.70.20	12	Normal	12	11	0.17	6	1	1.83	1.34	0
510.180.30.30	3	Normal	3	2	0.67	2	1	1.33	0.47	0
510.180.50.10	3	Normal	3	2	0.67	2	1	1.33	0.47	0
510.180.20	5	Normal	5	5	0.5	4	1	2	1.1	0.38
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
510.190.40	5	Normal	5	5	0.5	3	1	2	0.63	0
		Haircut	4	4	0.67	2	2	2	0	0
		MHCS	4	4	0.67	2	2	2	0	0
510.190.80	3	Normal	3	2	0.67	2	1	1.33	0.47	0
510.190.120	4	Normal	4	5	0.83	3	2	2.5	0.5	0.75
		Haircut	4	5	0.83	3	2	2.5	0.5	0.75
		MHCS	4	5	0.83	3	2	2.5	0.5	0.75
510.190.160.10	3	Normal	3	3	1	2	2	2	0	1
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
510.190.160.20	3	Normal	3	2	0.67	2	1	1.33	0.47	0
510.190.160.30	3	Normal	3	2	0.67	2	1	1.33	0.47	0
520.10.20	4	Normal	4	4	0.67	3	1	2	0.71	0.6
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
520.20	9	Normal	9	19	0.53	8	2	4.22	1.75	0.44
		Haircut	9	19	0.53	8	2	4.22	1.75	0.44
		MHCS	8	17	0.61	7	3	4.25	1.3	0.48
520.30	3	Normal	2	1	1	1	1	1	0	N/A

Table B.5: Statistics for MIPS complexes that induce a connected subgraph in the Y2H data. Statistics include MIPS complex ID, number of proteins in the complex, type of subgraph (either normal, haircut, or MHCS), number of proteins appearing in the Y2H data (n), and number of interactions in the Y2H data. Also gives the edge density (ED), degree statistics (max, min, average, and standard deviation), and clustering coefficient (CC).

Complex	Type	Ave. MCC	MCC Dev.	Max Bet.	Min. Bet.	Ave. Bet.	Bet. Dev.	Edge Con.	Vert. Con.
510.20	Normal	1	0	3	0	0.75	1.3	1	1
510.70.20	Normal	0.3	0.44	40	0	12.5	14.17	1	1
510.180.30.30	Normal	1	0	1	0	0.33	0.47	1	1
510.180.50.10	Normal	1	0	1	0	0.33	0.47	1	1
510.180.20	Normal	1	0	5	0	1	2	1	1
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
510.190.40	Normal	0.44	0.5	3.5	0	1.2	1.21	1	1
	Haircut	0.33	0.47	0.5	0.5	0.5	0	2	2
	MHCS	0.33	0.47	0.5	0.5	0.5	0	2	2
510.190.80	Normal	1	0	1	0	0.33	0.47	1	1
510.190.120	Normal	1	0	0.5	0	0.25	0.25	2	2
	Haircut	1	0	0.5	0	0.25	0.25	2	2
	MHCS	1	0	0.5	0	0.25	0.25	2	2
510.190.160.10	Normal	1	0	0	0	0	0	2	2
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
510.190.160.20	Normal	1	0	1	0	0.33	0.47	1	1
510.190.160.30	Normal	1	0	1	0	0.33	0.47	1	1
520.10.20	Normal	1	0	2	0	0.5	0.87	1	1
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
520.20	Normal	0.78	0.28	8.78	0	1.89	2.7	2	2
	Haircut	0.78	0.28	8.78	0	1.89	2.7	2	2
	MHCS	0.75	0.29	4.78	0.2	1.38	1.45	3	3
520.30	Normal	N/A	N/A	0	0	0	0	1	1

Table B.6: Statistics for MIPS complexes that induce a connected subgraph in the Y2H data. Statistics include MIPS complex ID, type of subgraph (either normal, haircut, or MHCS), mutual clustering coefficient statistics (average and standard deviation), betweenness statistics (max, min, average, and standard deviation), and edge and vertex connectivity.

B.2 Complexes with Some Interactions

For space reasons, statistics on complexes with MIPS ID numbers below 200 are in Tables B.7 and B.8. Those with IDs from 200 to 300 are in Tables B.9 and B.10. Those with IDs from 300 to 425 are in Tables B.11 and B.12. Those with IDs from 425 to 500 are in Tables B.13 and B.14. Those with IDs from 500 to 510.90 are in Tables B.15 and B.16. Complexes with IDs above 500.100 are in Tables B.17 and B.18.

Complex	Pro.	Type	n	m	ED	Max Deg.	Min Deg.	Ave. Deg.	Deg. Dev	CC
60	11	Normal	11	11	0.2	4	0	2	1.21	0.16
		Haircut	5	6	0.6	3	2	2.4	0.49	0.33
		MHCS	5	6	0.6	3	2	2.4	0.49	0.33
100	3	Normal	3	1	0.33	1	0	0.67	0.47	N/A
		MHCS	2	1	1	1	1	1	0	N/A
120.20	4	Normal	4	1	0.17	1	0	0.5	0.5	N/A
		MHCS	2	1	1	1	1	1	0	N/A
133.10	10	Normal	9	4	0.11	4	0	0.89	1.2	0
		MHCS	5	4	0.4	4	1	1.6	1.2	0
133.40	4	Normal	4	1	0.17	1	0	0.5	0.5	N/A
		MHCS	2	1	1	1	1	1	0	N/A
133.50	3	Normal	3	1	0.33	1	0	0.67	0.47	N/A
		MHCS	2	1	1	1	1	1	0	N/A
140.10.20	7	Normal	7	8	0.38	5	0	2.29	1.48	0.5
		Haircut	5	7	0.7	4	2	2.8	0.75	0.64
		MHCS	5	7	0.7	4	2	2.8	0.75	0.64
140.20.20	25	Normal	22	17	0.07	6	0	1.55	1.8	0.33
		Haircut	7	11	0.52	6	2	3.14	1.36	0.5
		MHCS	7	11	0.52	6	2	3.14	1.36	0.5
140.20.30	7	Normal	7	2	0.1	1	0	0.57	0.49	N/A
		MHCS	2	1	1	1	1	1	0	N/A
140.30.10	4	Normal	4	1	0.17	1	0	0.5	0.5	N/A
		MHCS	2	1	1	1	1	1	0	N/A
140.30.20	14	Normal	13	5	0.06	2	0	0.77	0.8	1
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
140.30.30.10	8	Normal	8	1	0.04	1	0	0.25	0.43	N/A
		MHCS	2	1	1	1	1	1	0	N/A
160	7	Normal	7	3	0.14	2	0	0.86	0.83	0
		MHCS	4	3	0.5	2	1	1.5	0.5	0
177	5	Normal	3	1	0.33	1	0	0.67	0.47	N/A
		MHCS	2	1	1	1	1	1	0	N/A

Table B.7: Statistics for MIPS complexes that do not induce a connected subgraph in the Y2H data but have some interactions between their proteins. Statistics include MIPS complex ID, number of proteins in the complex, type of subgraph (either normal, haircut, or MHCS), number of proteins appearing in the Y2H data (n), and number of interactions in the Y2H data. Also gives the edge density (ED), degree statistics (max, min, average, and standard deviation), and clustering coefficient (CC).

Complex	Type	Ave. MCC	MCC Dev.	Max Bet.	Min. Bet.	Ave. Bet.	Bet. Dev	Edge Con.	Vert. Con.
60	Normal	0.3	0.4	21.5	0	5.64	7.69	0	0
	Haircut	0.55	0.42	1.5	0	0.8	0.6	2	2
	MHCS	0.55	0.42	1.5	0	0.8	0.6	2	2
100	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
120.20	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
133.10	Normal	1	0	6	0	0.67	1.89	0	0
	MHCS	1	0	6	0	1.2	2.4	1	1
133.40	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
133.50	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
140.10.20	Normal	0.93	0.17	6	0	1	2.05	0	0
	Haircut	0.9	0.2	2	0	0.6	0.73	2	2
	MHCS	0.9	0.2	2	0	0.6	0.73	2	2
140.20.20	Normal	0.34	0.43	27	0	3.77	7.54	0	0
	Haircut	0.82	0.24	7.5	0	1.43	2.53	2	2
	MHCS	0.82	0.24	7.5	0	1.43	2.53	2	2
140.20.30	Normal	0	0	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
140.30.10	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
140.30.20	Normal	0.16	0.36	0	0	0	0	0	0
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
140.30.30.10	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
160	Normal	0.5	0.5	2	0	0.57	0.9	0	0
	MHCS	0.5	0.5	2	0	1	1	1	1
177	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1

Table B.8: Statistics for MIPS complexes that do not induce a connected subgraph in the Y2H data but have some interactions between their proteins. Statistics include MIPS complex ID, type of subgraph (either normal, haircut, or MHCS), mutual clustering coefficient statistics (average and standard deviation), betweenness statistics (max, min, average, and standard deviation), and edge and vertex connectivity.

Complex	Pro.	Type	n	m	ED	Max Deg.	Min Deg.	Ave. Deg.	Deg. Dev.	CC
220	15	Normal	12	5	0.08	3	0	0.83	0.8	0
		MHCS	4	3	0.5	3	1	1.5	0.87	0
230.20.10	6	Normal	5	2	0.2	2	0	0.8	0.75	0
		MHCS	3	2	0.67	2	1	1.33	0.47	0
230.20.20	16	Normal	15	8	0.08	3	0	1.07	0.85	0
		MHCS	5	4	0.4	3	1	1.6	0.8	0
260.20.30	4	Normal	3	1	0.33	1	0	0.67	0.47	N/A
		MHCS	2	1	1	1	1	1	0	N/A
260.20.40	8	Normal	8	12	0.43	5	0	3	1.5	0.64
		Haircut	7	12	0.57	5	2	3.43	1.05	0.64
		MHCS	5	9	0.9	4	3	3.6	0.49	0.88
260.30.10	8	Normal	5	1	0.1	1	0	0.4	0.49	N/A
		MHCS	2	1	1	1	1	1	0	N/A
260.30.20	11	Normal	9	7	0.19	3	0	1.56	0.83	0.43
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
260.50.10	8	Normal	7	1	0.05	1	0	0.29	0.45	N/A
		MHCS	2	1	1	1	1	1	0	N/A
260.60	10	Normal	10	15	0.33	5	0	3	1.79	0.85
		Haircut	6	13	0.87	5	3	4.33	0.75	0.87
		MHCS	5	10	1	4	4	4	0	1
260.80	4	Normal	3	1	0.33	1	0	0.67	0.47	N/A
		MHCS	2	1	1	1	1	1	0	N/A
260.90	6	Normal	5	1	0.1	1	0	0.4	0.49	N/A
		MHCS	2	1	1	1	1	1	0	N/A
270.10.10	4	Normal	4	2	0.33	2	0	1	0.71	0
		MHCS	3	2	0.67	2	1	1.33	0.47	0
290.10	9	Normal	6	3	0.2	2	0	1	0.82	0
		MHCS	4	3	0.5	2	1	1.5	0.5	0

Table B.9: Statistics for MIPS complexes that do not induce a connected subgraph in the Y2H data but have some interactions between their proteins. Statistics include MIPS complex ID, number of proteins in the complex, type of subgraph (either normal, haircut, or MHCS), number of proteins appearing in the Y2H data (n), and number of interactions in the Y2H data. Also gives the edge density (ED), degree statistics (max, min, average, and standard deviation), and clustering coefficient (CC).

Complex	Type	Ave. MCC	MCC Dev.	Max Bet.	Min. Bet.	Ave. Bet.	Bet. Dev	Edge Con.	Vert. Con.
220	Normal	0.13	0.34	3	0	0.25	0.83	0	0
	MHCS	1	0	3	0	0.75	1.3	1	1
230.20.10	Normal	1	0	1	0	0.2	0.4	0	0
	MHCS	1	0	1	0	0.33	0.47	1	1
230.20.20	Normal	0.12	0.33	5	0	0.8	1.47	0	0
	MHCS	0.57	0.49	5	0	1.6	2.06	1	1
260.20.30	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
260.20.40	Normal	0.64	0.37	4.33	0	1.25	1.51	0	0
	Haircut	0.64	0.37	4.33	0	1.43	1.54	2	2
	MHCS	1	0	0.33	0	0.2	0.16	3	3
260.30.10	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
260.30.20	Normal	0.24	0.4	4	0	0.89	1.45	0	0
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
260.50.10	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
260.60	Normal	0.47	0.5	1	0	0.3	0.38	0	0
	Haircut	1	0	0.67	0	0.33	0.33	3	3
	MHCS	1	0	0	0	0	0	4	4
260.80	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
260.90	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
270.10.10	Normal	1	0	1	0	0.25	0.43	0	0
	MHCS	1	0	1	0	0.33	0.47	1	1
290.10	Normal	0.5	0.5	2	0	0.67	0.94	0	0
	MHCS	0.5	0.5	2	0	1	1	1	1

Table B.10: Statistics for MIPS complexes that do not induce a connected subgraph in the Y2H data but have some interactions between their proteins. Statistics include MIPS complex ID, type of subgraph (either normal, haircut, or MHCS), mutual clustering coefficient statistics (average and standard deviation), betweenness statistics (max, min, average, and standard deviation), and edge and vertex connectivity.

Complex	Pro.	Type	n	m	ED	Max Deg.	Min Deg.	Ave. Deg.	Deg. Dev	CC
310	13	Normal	12	2	0.03	2	0	0.33	0.62	0
		MHCS	3	2	0.67	2	1	1.33	0.47	0
310.10	4	Normal	4	2	0.33	2	0	1	0.71	0
		MHCS	3	2	0.67	2	1	1.33	0.47	0
310.40	6	Normal	5	1	0.1	1	0	0.4	0.49	N/A
		MHCS	2	1	1	1	1	1	0	N/A
360.10.10	15	Normal	15	13	0.12	4	0	1.73	1.06	0
		Haircut	4	4	0.67	2	2	2	0	0
		MHCS	4	4	0.67	2	2	2	0	0
360.10.20	18	Normal	16	18	0.15	5	0	2.25	1.56	0.43
		Haircut	8	12	0.43	4	2	3	0.71	0.69
		MHCS	4	6	1	3	3	3	0	1
400	10	Normal	10	3	0.07	2	0	0.6	0.8	0
		MHCS	4	3	0.5	2	1	1.5	0.5	0
410.30	16	Normal	15	16	0.15	4	0	2.13	1.15	0.64
		Haircut	8	11	0.39	4	2	2.75	0.83	0.82
		MHCS	4	6	1	3	3	3	0	1
410.35	20	Normal	18	18	0.12	4	0	2	1.2	0.68
		Haircut	11	14	0.25	4	2	2.55	0.78	0.84
		MHCS	4	6	1	3	3	3	0	1
410.40.30	5	Normal	3	1	0.33	1	0	0.67	0.47	N/A
		MHCS	2	1	1	1	1	1	0	N/A
420.30	10	Normal	6	1	0.07	1	0	0.33	0.47	N/A
		MHCS	2	1	1	1	1	1	0	N/A

Table B.11: Statistics for MIPS complexes that do not induce a connected subgraph in the Y2H data but have some interactions between their proteins. Statistics include MIPS complex ID, number of proteins in the complex, type of subgraph (either normal, haircut, or MHCS), number of proteins appearing in the Y2H data (n), and number of interactions in the Y2H data. Also gives the edge density (ED), degree statistics (max, min, average, and standard deviation), and clustering coefficient (CC).

Complex	Type	Ave. MCC	MCC Dev.	Max Bet.	Min Bet.	Ave. Bet.	Bet. Dev.	Edge Con.	Vert. Con.
310	Normal	1	0	1	0	0.08	0.28	0	0
	MHCS	1	0	1	0	0.33	0.47	1	1
310.10	Normal	1	0	1	0	0.25	0.43	0	0
	MHCS	1	0	1	0	0.33	0.47	1	1
310.40	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
360.10.10	Normal	0.15	0.31	45.5	0	12.8	13.56	0	0
	Haircut	0.33	0.47	0.5	0.5	0.5	0	2	2
	MHCS	0.33	0.47	0.5	0.5	0.5	0	2	2
360.10.20	Normal	0.31	0.42	47	0	10.19	15.37	0	0
	Haircut	0.51	0.45	12.5	0	3.13	5.27	1	1
	MHCS	1	0	0	0	0	0	3	3
400	Normal	0.5	0.5	2	0	0.4	0.8	0	0
	MHCS	0.5	0.5	2	0	1	1	1	1
410.30	Normal	0.18	0.36	6	0	1.6	2.22	0	0
	Haircut	0.46	0.5	1	0	0.25	0.43	0	0
	MHCS	1	0	0	0	0	0	3	3
410.35	Normal	0.18	0.37	4	0	0.67	1.15	0	0
	Haircut	0.29	0.45	1	0	0.18	0.39	0	0
	MHCS	1	0	0	0	0	0	3	3
410.40.30	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
420.30	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1

Table B.12: Statistics for MIPS complexes that do not induce a connected subgraph in the Y2H data but have some interactions between their proteins. Statistics include MIPS complex ID, type of subgraph (either normal, haircut, or MHCS), mutual clustering coefficient statistics (average and standard deviation), betweenness statistics (max, min, average, and standard deviation), and edge and vertex connectivity.

Complex	Pro.	Type	n	m	ED	Max Deg.	Min Deg.	Ave. Deg.	Deg. Dev	CC
440.12.10	7	Normal	7	5	0.24	2	0	1.43	0.73	0
		MHCS	6	5	0.33	2	1	1.67	0.47	0
440.30.10	37	Normal	29	16	0.04	5	0	1.1	1.32	0.11
		Haircut	6	7	0.47	4	2	2.33	0.75	0.27
		MHCS	4	4	0.67	2	2	2	0	0
440.30.10.10	3	Normal	3	1	0.33	1	0	0.67	0.47	N/A
		MHCS	2	1	1	1	1	1	0	N/A
440.30.20	24	Normal	18	4	0.03	2	0	0.44	0.68	0
		MHCS	4	3	0.5	2	1	1.5	0.5	0
440.30.30	11	Normal	7	2	0.1	1	0	0.57	0.49	N/A
		MHCS	2	1	1	1	1	1	0	N/A
440.40	13	Normal	7	1	0.05	1	0	0.29	0.45	N/A
		MHCS	2	1	1	1	1	1	0	N/A
445.10	5	Normal	5	4	0.4	3	0	1.6	1.02	0.6
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
445.20	4	Normal	4	2	0.33	2	0	1	0.71	0
		MHCS	3	2	0.67	2	1	1.33	0.47	0
445.30	4	Normal	4	3	0.5	2	0	1.5	0.87	1
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
475.05	6	Normal	6	1	0.07	1	0	0.33	0.47	N/A
		MHCS	2	1	1	1	1	1	0	N/A
480.10	13	Normal	13	11	0.14	3	0	1.69	0.99	0.21
		Haircut	8	9	0.32	3	2	2.25	0.43	0.25
		MHCS	8	9	0.32	3	2	2.25	0.43	0.25
480.20	14	Normal	14	11	0.12	4	0	1.57	1.29	0.33
		Haircut	4	5	0.83	3	2	2.5	0.5	0.75
		MHCS	4	5	0.83	3	2	2.5	0.5	0.75
490	5	Normal	5	2	0.2	2	0	0.8	0.75	0
		MHCS	3	2	0.67	2	1	1.33	0.47	0

Table B.13: Statistics for MIPS complexes that do not induce a connected subgraph in the Y2H data but have some interactions between their proteins. Statistics include MIPS complex ID, number of proteins in the complex, type of subgraph (either normal, haircut, or MHCS), number of proteins appearing in the Y2H data (n), and number of interactions in the Y2H data. Also gives the edge density (ED), degree statistics (max, min, average, and standard deviation), and clustering coefficient (CC).

Complex	Type	Ave. MCC	MCC Dev.	Max Bet.	Min Bet.	Ave. Bet.	Bet. Dev.	Edge Con.	Vert. Con.
440.12.10	Normal	0.23	0.37	6	0	2.86	2.59	0	0
	MHCS	0.23	0.37	6	0	3.33	2.49	1	1
440.30.10	Normal	0.15	0.32	46.5	0	4.41	10.71	0	0
	Haircut	0.47	0.43	6.5	0	1.67	2.25	2	1
	MHCS	0.33	0.47	0.5	0.5	0.5	0	2	2
440.30.10.10	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
440.30.20	Normal	0.17	0.37	2	0	0.22	0.63	0	0
	MHCS	0.5	0.5	2	0	1	1	1	1
440.30.30	Normal	0	0	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
440.40	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
445.10	Normal	1	0	2	0	0.4	0.8	0	0
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
445.20	Normal	1	0	1	0	0.25	0.43	0	0
	MHCS	1	0	1	0	0.33	0.47	1	1
445.30	Normal	1	0	0	0	0	0	0	0
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
475.05	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
480.10	Normal	0.15	0.27	8	0	3.54	3.13	0	0
	Haircut	0.25	0.31	5.5	0	3.75	1.66	2	2
	MHCS	0.25	0.31	5.5	0	3.75	1.66	2	2
480.20	Normal	0.25	0.35	18	0	5.57	6.64	0	0
	Haircut	1	0	0.5	0	0.25	0.25	2	2
	MHCS	1	0	0.5	0	0.25	0.25	2	2
490	Normal	1	0	1	0	0.2	0.4	0	0
	MHCS	1	0	1	0	0.33	0.47	1	1

Table B.14: Statistics for MIPS complexes that do not induce a connected subgraph in the Y2H data but have some interactions between their proteins. Statistics include MIPS complex ID, type of subgraph (either normal, haircut, or MHCS), mutual clustering coefficient statistics (average and standard deviation), betweenness statistics (max, min, average, and standard deviation), and edge and vertex connectivity.

Complex	Pro.	Type	n	m	ED	Max Deg.	Min Deg.	Ave. Deg.	Deg. Dev	CC
500.10.30	5	Normal	4	2	0.33	2	0	1	0.71	0
		MHCS	3	2	0.67	2	1	1.33	0.47	0
500.10.40	7	Normal	7	4	0.19	2	0	1.14	0.83	1
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
500.20.10	6	Normal	4	1	0.17	1	0	0.5	0.5	N/A
		MHCS	2	1	1	1	1	1	0	N/A
500.40.10	81	Normal	20	2	0.01	2	0	0.2	0.51	0
		MHCS	3	2	0.67	2	1	1.33	0.47	0
500.50	13	Normal	6	5	0.33	4	0	1.67	1.25	0.38
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
510.10	14	Normal	14	9	0.1	5	0	1.29	1.16	0.25
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
510.30	3	Normal	3	1	0.33	1	0	0.67	0.47	N/A
		MHCS	2	1	1	1	1	1	0	N/A
510.40.10	13	Normal	13	8	0.1	4	0	1.23	0.97	0
		MHCS	6	5	0.33	4	1	1.67	1.11	0
510.40.20	21	Normal	19	24	0.14	6	0	2.53	1.87	0.26
		Haircut	10	17	0.38	6	2	3.4	1.28	0.37
		MHCS	4	6	1	3	3	3	0	1
510.90	3	Normal	3	1	0.33	1	0	0.67	0.47	N/A
		MHCS	2	1	1	1	1	1	0	N/A

Table B.15: Statistics for MIPS complexes that do not induce a connected subgraph in the Y2H data but have some interactions between their proteins. Statistics include MIPS complex ID, number of proteins in the complex, type of subgraph (either normal, haircut, or MHCS), number of proteins appearing in the Y2H data (n), and number of interactions in the Y2H data. Also gives the edge density (ED), degree statistics (max, min, average, and standard deviation), and clustering coefficient (CC).

Complex	Type	Ave. MCC	MCC Dev.	Max Bet.	Min Bet.	Ave. Bet.	Bet. Dev.	Edge Con.	Vert. Con.
500.10.30	Normal	1	0	1	0	0.25	0.43	0	0
	MHCS	1	0	1	0	0.33	0.47	1	1
500.10.40	Normal	0.33	0.47	0	0	0	0	0	0
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
500.20.10	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
500.40.10	Normal	1	0	1	0	0.05	0.22	0	0
	MHCS	1	0	1	0	0.33	0.47	1	1
500.50	Normal	1	0	5	0	0.83	1.86	0	0
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
510.10	Normal	0.2	0.4	9	0	0.64	2.32	0	0
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
510.30	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
510.40.10	Normal	0.17	0.37	9	0	1.08	2.53	0	0
	MHCS	0.64	0.48	9	0	2.17	3.39	1	1
510.40.20	Normal	0.27	0.39	48	0	8.63	13.29	0	0
	Haircut	0.46	0.33	10	0	3.2	3.63	2	2
	MHCS	1	0	0	0	0	0	3	3
510.90	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1

Table B.16: Statistics for MIPS complexes that do not induce a connected subgraph in the Y2H data but have some interactions between their proteins. Statistics include MIPS complex ID, type of subgraph (either normal, haircut, or MHCS), mutual clustering coefficient statistics (average and standard deviation), betweenness statistics (max, min, average, and standard deviation), and edge and vertex connectivity.

Complex	Pro.	Type	n	m	ED	Max Deg.	Min Deg.	Ave. Deg.	Deg. Dev	CC
510.100	9	Normal	9	6	0.17	3	0	1.33	0.94	0.5
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
510.110	3	Normal	3	1	0.33	1	0	0.67	0.47	N/A
		MHCS	2	1	1	1	1	1	0	N/A
510.120	13	Normal	13	5	0.06	2	0	0.77	0.7	0
		MHCS	3	2	0.67	2	1	1.33	0.47	0
510.140	3	Normal	3	1	0.33	1	0	0.67	0.47	N/A
		MHCS	2	1	1	1	1	1	0	N/A
510.150	5	Normal	4	2	0.33	2	0	1	0.71	0
		MHCS	3	2	0.67	2	1	1.33	0.47	0
510.160	4	Normal	4	3	0.5	2	0	1.5	0.87	1
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
510.180.10.10	3	Normal	3	1	0.33	1	0	0.67	0.47	N/A
		MHCS	2	1	1	1	1	1	0	N/A
510.180.10.30	9	Normal	9	5	0.14	3	0	1.11	0.99	0.6
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
510.190.10.10	6	Normal	5	2	0.2	2	0	0.8	0.75	0
		MHCS	3	2	0.67	2	1	1.33	0.47	0
510.190.10.20.1016		Normal	15	8	0.08	3	0	1.07	0.85	0
		MHCS	5	4	0.4	3	1	1.6	0.8	0
510.190.50	10	Normal	9	5	0.14	4	0	1.11	1.29	0.38
		Haircut	3	3	1	2	2	2	0	1
		MHCS	3	3	1	2	2	2	0	1
510.190.110	13	Normal	13	19	0.24	6	0	2.92	1.9	0.35
		Haircut	10	17	0.38	6	2	3.4	1.43	0.41
		MHCS	6	11	0.73	4	3	3.67	0.47	0.6
510.190.150	4	Normal	4	2	0.33	2	0	1	0.71	0
		MHCS	3	2	0.67	2	1	1.33	0.47	0

Table B.17: Statistics for MIPS complexes that do not induce a connected subgraph in the Y2H data but have some interactions between their proteins. Statistics include MIPS complex ID, number of proteins in the complex, type of subgraph (either normal, haircut, or MHCS), number of proteins appearing in the Y2H data (n), and number of interactions in the Y2H data. Also gives the edge density (ED), degree statistics (max, min, average, and standard deviation), and clustering coefficient (CC).

Complex	Type	Ave. MCC	MCC Dev.	Max Bet.	Min Bet.	Ave. Bet.	Bet. Dev.	Edge Con.	Vert. Con.
510.100	Normal	0.33	0.47	2	0	0.33	0.67	0	0
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
510.110	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
510.120	Normal	0.09	0.28	1	0	0.15	0.36	0	0
	MHCS	1	0	1	0	0.33	0.47	1	1
510.140	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
510.150	Normal	1	0	1	0	0.25	0.43	0	0
	MHCS	1	0	1	0	0.33	0.47	1	1
510.160	Normal	1	0	0	0	0	0	0	0
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
510.180.10.10	Normal	N/A	N/A	0	0	0	0	0	0
	MHCS	N/A	N/A	0	0	0	0	1	1
510.180.10.30	Normal	0.38	0.49	2	0	0.22	0.63	0	0
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
510.190.10.10	Normal	1	0	1	0	0.2	0.4	0	0
	MHCS	1	0	1	0	0.33	0.47	1	1
510.190.10.20.10	Normal	0.12	0.33	5	0	0.8	1.47	0	0
	MHCS	0.57	0.49	5	0	1.6	2.06	1	1
510.190.50	Normal	1	0	5	0	0.56	1.57	0	0
	Haircut	1	0	0	0	0	0	2	2
	MHCS	1	0	0	0	0	0	2	2
510.190.110	Normal	0.47	0.41	26.33	0	5.46	7.83	0	0
	Haircut	0.48	0.37	19.67	0	4	5.53	2	1
	MHCS	0.73	0.2	1.17	0.25	0.67	0.38	3	3
510.190.150	Normal	1	0	1	0	0.25	0.43	0	0
	MHCS	1	0	1	0	0.33	0.47	1	1

Table B.18: Statistics for MIPS complexes that do not induce a connected subgraph in the Y2H data but have some interactions between their proteins. Statistics include MIPS complex ID, type of subgraph (either normal, haircut, or MHCS), mutual clustering coefficient statistics (average and standard deviation), betweenness statistics (max, min, average, and standard deviation), and edge and vertex connectivity.

B.3 Complexes with no interactions

Complex	Pro.	n	Complex	Pro.	n
20	3	3	410.40.60	4	3
120.10	4	3	410.50	4	1
130	8	4	420.20	4	1
140.30.30.20	3	2	420.40	11	1
180.30	3	1	420.50	18	6
190.10	3	1	420.60	14	0
200	4	1	430	4	3
210	4	3	440.10.20	4	3
240.20	3	3	440.10.30	3	2
260.20.20	4	2	440.20	4	1
260.30.30.10	3	1	440.30	3	3
260.50.20	8	6	475.10	3	3
270.10	3	3	500.10.20	3	2
290.20.10	5	1	500.10.80	3	3
320	8	5	500.10.110	3	3
330	3	1	500.40.20	57	14
360.10	3	3	500.60	4	1
390	5	3	500.60.10	44	5
410.40.20	3	3	500.60.20	31	9

Table B.19: MIPS complexes that have no interactions between their proteins in Y2H data. Statistics include MIPS complex ID, number of proteins in the complex, and number of proteins appearing in the Y2H data (n).