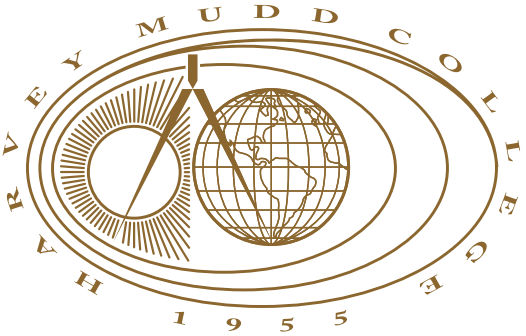
2012

Detecting Covert Members of Terrorist Networks

Alice Paul
Harvey Mudd College



Detecting Covert Members of Terrorist Networks

Alice Paul

Susan Martonosi, Advisor

Nicholas Pippenger, Reader

May, 2012

HARVEY MUDD
COLLEGE
Department of Mathematics

Abstract

Terrorism threatens both international peace and security and is a national concern. It is believed that terrorist organizations rely heavily on a few key leaders and that destroying such an organization's leadership is essential to reducing its influence. Martonosi et al. (2011) argues that increasing the amount of communication through a key leader increases the likelihood of detection. If we model a covert organization as a social network where edges represent communication between members, we want to determine the subset of members to remove that maximizes the amount of communication through the key leader. A mixed-integer linear program representing this problem is presented as well as a decomposition for this optimization problem. As these approaches prove impractical for larger graphs, often running out of memory, the last section focuses on structural characteristics of vertices and subsets that increase communication. Future work should develop these structural properties as well as heuristics for solving this problem.

Contents

Abstract **iii**

Acknowledgments **xi**

1 Introduction **1**

2 Literature Review **3**
2.1 Previous Network Interdiction Research 3
2.2 A New Model 4

3 Optimization Framework **7**
3.1 Max-Flow Subproblems 7
3.2 Choosing a Subset 10
3.3 Implementation 12

4 Decomposition **15**
4.1 Bender's Decomposition 15
4.2 Delayed Column Generation 17
4.3 Application 17
4.4 Implementation 19

5 Structural Properties **21**
5.1 Previous Results on Single-LOMAX 21
5.2 LOMAX 22
5.3 Examining Cyclic Structures 25
5.4 Pairs of Vertices 35

6 Conclusions and Future Work **43**

Bibliography **45**

List of Figures

5.1 An example of how a graph can be simplified by combining vertices with a single edge-disjoint path to k 25
5.2 Examples of chordless cycles 26
5.3 An example of a graph G that has a single cycle 27
5.4 An example of a graph G with p parallel paths between k and a vertex u 30
5.5 An example of a graph G where k is in a single chordless cycle C 33
5.6 An example of calculating the number of edge-disjoint paths between vertices in a graph G where k is in a single chordless cycle C 33
5.7 An example of a stack of cycles 35
5.8 An example in which removing vertices in separate components is not beneficial 37
5.9 An example in which the list of cycles that contain u is a subset of the cycles that contain v 37
5.10 An example in which all cycles with v and k contain u 38
5.11 An example in which removing $\{u, v\}$ increases the number of edge-disjoint paths between s and t that go through k, but removing u or v alone does not 39

List of Tables

3.1 Run times and results of the MIP representing LOMAX on graphs with 20–30 vertices 13

Acknowledgments

First, I would like to thank my advisor, Professor Susan Martonosi, for her encouragement, insight, and advice throughout this project and my entire career at Harvey Mudd College. Additionally, I would like to thank the entire Harvey Mudd College Math Department for providing me with the opportunities in the classroom and outside that helped me grow as a mathematician.

Lastly, I would like to thank my friends and family for their unending and much appreciated support.

Chapter 1

Introduction

Terrorism threatens both international peace and security and has become a national concern, especially after September 11, 2001. It is believed that terrorist organizations rely heavily on a few key leaders and that destroying such an organization's leadership is essential to disrupting its influence. However these leaders are difficult to detect by intelligence agencies (Martonosi and Altner, 2009).

Terrorist organizations can be modeled as social networks where vertices represent members of the organization and edges represent direct communication between members. Terrorist leaders may choose to avoid being involved in direct communication to evade detection. Martonosi et al. (2011) argue that increasing the amount of communication through a key vertex increases that member's visibility to intelligence agencies. A question emerges: Which vertices can be removed to increase the visibility of leaders in a terrorist network?

In this paper, terrorist networks will be represented as graphs where vertices represent members and edges represent communication between members. The leader of the organization is marked as the key vertex k. In the model used, the amount of communication in the network is interpreted as the total number of edge-disjoint paths between pairs of members, not including k, since these paths represent possible independent chains of communication. We are interested in the amount of communication k participates in, which can be measured by the number of paths that must contain k. As argued in Martonosi et al. (2011), this metric can be interpreted as the importance of k in maintaining communication in the graph. The more important k is to maintaining communication, the more edge-disjoint paths must pass through k.

Removing members, or vertices, is one way of altering the communication flow in the network and possibly increasing k's activity. To determine the subset of vertices to remove that maximizes communication through k, the problem is translated into a mixed-integer linear program where the objective function represents the communication that must pass through k and the optimal solution is the subset whose removal maximizes this communication.

Since solving the mixed-integer program on even small graphs (thirty vertices or more) proved computationally intractable, a decomposition algorithm was implemented to decrease the computational complexity. Then, structural properties were investigated to determine necessary conditions for subsets that increase k's activity. This also helps reduce the complexity of the problem by eliminating subsets from consideration and could prove useful in a heuristic approach to the problem.

In Chapter 2, we review previous research on network disruption and covert networks. Past work by Martonosi et al. (2011) presents a modeling framework for this problem upon which this paper will build. Then in Chapter 3, we present the formulation of our mixed-integer linear program and results. A decomposition of this formulation and results are discussed in Chapter 4. As the results show that the decomposition remains impractical, Chapter 5 determines structural characteristics of subsets (and vertices within these subsets) that increase communication through k. In particular, we look at vertices within certain graph structures and pairs of vertices. Future work should focus on these structural characteristics in order to simplify the given problem and develop more practical algorithms. Conclusions for this thesis and plans for future work are discussed in Chapter 6.

Chapter 2

Literature Review

The goal of research in terrorist networks is to learn new ways of disrupting these covert networks. In this chapter we will review previous literature on network interdiction and a model of disruption that focuses on the communication through a key member of the network.

2.1 Previous Network Interdiction Research

Previous work involving network interdiction has focused on disconnecting the network or increasing the length of the shortest path within a network. This type of interdiction limits or destroys the flow of resources within the graph and is useful for studying systems such as power grids, the internet, and military operations (Royset and Wood, 2007; Martonosi et al., 2011).

However, two main problems arise when applying these previous results to terrorist networks. First, in disrupting terrorist networks, we may not have the resources necessary to disconnect the network. Instead, targeting a leader in the terrorist organization could have an equally disruptive effect for lower cost and chance of detection Martonosi et al. (2011). Additionally, focusing on lengthening the longest path is not as applicable since covert organizations tend to already communicate along long paths (Sageman, 2004).

Second, these models focus on graphs that exhibit a high clustering coefficient and an average path length around that of a random graph (Li et al., 2005). Therefore, the vertices targeted for removal often are of high degree and have a high amount of flow through them (Motter and Lai, 2002; Martonosi et al., 2011). However, terrorist networks have exhibited a longer

average path length, which makes this work less applicable (Martonosi et al., 2011).

Another area of research focuses on cascade-based attacks. Given a network with capacities on the vertices, a cascade-based attack tries to remove some vertices in order to cause the failure of the largest subset of vertices by increasing some value on the vertices beyond capacity (Motter and Lai, 2002). Here, the capacity of vertices could be interpreted as a level of communication over which we could detect that vertex. However, in our problem we are trying to optimally remove vertices in order to increase the communication through a single key vertex, which most likely has a lower amount of communication. Thus, a cascade-based attack, which focuses on the largest subset being detected, would most likely target other vertices, making this approach irrelevant.

Instead, we want a network disruption model that accounts for the unique structure of terrorist networks and focuses on increasing the communication through a single vertex.

2.2 A New Model

Martonosi et al. (2011) have presented a new model for network interdiction that focuses on a key vertex and the communication through that vertex. We define a few metrics below as defined by Martonosi et al. (2011).

Consider an undirected graph $G = (V, E)$ with key vertex k.

Definition 2.1. *(Martonosi et al., 2011) Let $z_{s,t}(G)$ be the number of edge-disjoint $s-t$ paths in graph G. The **flow capacity of graph G with respect to vertex k** is defined as*

$$Z_k(G) = \sum_{\substack{s,t \in V \setminus \{k\} \\ s \neq t}} z_{s,t}(G).$$

The flow capacity of G measures the total amount of flow or communication through the network that does not start or end at k.

In a max-flow problem with integer edge-capacities, unit values of flow are pushed between s and t along paths between the two vertices. Since each edge has capacity one, an edge can only be used once, and the maximum flow can be represented as edge-disjoint paths. Therefore, the total communication can be computed as a sum of max-flow problems over all pairs of vertices.

Since the objective is to detect the leader k, we want to increase the amount of communication through that vertex. If our total communication

is represented as the number of all-pairs s-t paths, then the amount of communication k must participate in is the number of edge-disjoint paths that must contain k. We call this the load of k

Definition 2.2. *(Martonosi et al., 2011) The **load of a vertex** k **in graph** G is defined as*

$$\mathcal{L}_k(G) = Z_k(G) - Z_k(G \setminus \{k\}).$$

That is, the load of k is the change in flow capacity after k is removed. This measures how important vertex k is to maintaining the flow in the graph. We can also measure how removing a subset of vertices affects the load of k

Definition 2.3. *(Martonosi et al., 2011) The **load effect of subset** S **on key vertex** k is the change in the key vertex k's load when the subset is removed*

$$\mathcal{E}_k(G, S) = \mathcal{L}_k(G \setminus S) - \mathcal{L}_k(G).$$

A positive load effect indicates the load of k will increase if subset S is removed, which is desirable since it increases the amount of communication through k. Therefore, the problem becomes identifying the subset of vertices with the greatest positive load effect.

There are two cases addressed in Martonosi et al. (2011). The *Load Maximization Problem* (LOMAX), which addresses finding the subset of vertices with the maximum load effect, and the *Single Vertex Deletion Load Maximization Problem* (Single-LOMAX), a special case in which a single vertex is removed. We address the previous results of Martonosi et al. (2011) in both these areas in the sections following.

2.2.1 Single-LOMAX

Finding a single vertex that maximizes the load of k is a special case of LOMAX. As mentioned above, the load of k in a graph G can be computed using a linear max-flow network problem (See Chapter 3 for details). Solving a max-flow problem takes polynomial time. Therefore, for every vertex u in G, we can compute the load effect of u on k in polynomial time, and we can determine the vertex with the highest load effect to remove in polynomial time, as well.

Martonosi et al. (2011) observed empirically that for a key vertex that exhibits the highest average betweenness, closeness, and degree centrality, a vertex with positive load effect on the key vertex will almost always exist. An example in which such a vertex does not exist is if the graph is a cycle on

n vertices or any graph where the key vertex is a leaf. However, the number of vertices with a significant load effect is generally low since deleting a vertex reduces the overall flow in the graph, and therefore the load of k, in most cases.

The structural characteristics of vertices that have a positive load effect on the key vertex are of interest since knowing these properties could help find vertices more efficiently and possibly have extensions for LOMAX.

Martonosi et al. (2011) have proven several properties of vertices that cannot have a positive load effect, discussed further in Chapter 5. These results helped to create a heuristic algorithm that eliminates vertices that break these conditions. The resulting algorithm has the potential to decrease the computation time for identifying positive load effect vertices. Another algorithm using divide-and-conquer was also tested. However, characterizing vertices that *do* increase the load remains to be considered (see Chapter 5).

2.2.2 LOMAX

The Load Maximization Problem is computationally more difficult. Since there are $O(2^n)$ possible subsets that we could consider, we can no longer solve for the best subset using brute force in polynomial time.

Martonosi et al. (2011) argued that LOMAX could not be modeled as an integer linear programming problem. Instead, they develop a genetic algorithm on initial subsets of vertices. The weakness of a genetic algorithm is that it is not guaranteed to reach an optimal or near-optimal solution. This algorithm outperforms random search, but needs improvements to be a viable option for solving this problem.

As shown in Chapter 3, it is possible to model LOMAX as a mixed-integer linear program. However, this MIP is computationally intractable for graphs with more than thirty vertices.

Chapter 3

Optimization Framework

The LOMAX problem tries to determine a subset of vertices whose removal maximizes the load effect on a given vertex k. This can be rephrased as determining the subset of vertices S that maximizes the load of k in $G \setminus S$, where the load of k is the number of edge-disjoint s-t paths ($s \neq t$ and $s, t \neq k$) that must include k.

To formulate LOMAX as an optimization problem, we first formulate a linear program to solve for the load of k in any graph G. We can do this by looking at two max-flow subproblems.

3.1 Max-Flow Subproblems

The load of k in a graph $G = (V, E)$ is the difference between two values of Z_k

$$\mathcal{L}_k(G) = Z_k(G) - Z_k(G \setminus \{k\}) \tag{3.1}$$

Each Z_k is the number of edge-disjoint s-t paths over all pairs (s, t), s and t not equal to k.

For each pair (s, t), the number of edge-disjoint paths is computed by solving a max-flow problem between s and t where the edge capacities are one. Given a pair (s, t), the max-flow optimization problem that computes the number of edge-disjoint s-t paths over a set of vertices V and edges E can be written as

$$\begin{aligned}
&\text{Maximize} \quad v_{s,t} \\
&\text{subject to} \\
&\qquad \sum_{j:(i,j)\in E} x_{i,j,s,t} - \sum_{k:(k,i)\in E} x_{k,i,s,t} = \begin{cases} v_{s,t} & \text{if } i = s \\ -v_{s,t} & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \\
&\qquad \text{for all } i \in V \\
&\qquad x_{i,j,s,t} \leq 1 \text{ for all } (i,j) \in E \\
&\qquad v_{s,t} \geq 0 \\
&\qquad x_{i,j,s,t} \geq 0 \text{ for all } (i,j) \in E.
\end{aligned} \tag{3.2}$$

Computing the number of paths between different s-t pairs can be done independently. Therefore, we can create an optimization program that computes all s-t pair paths in a single maximization problem. Let V_{-k} be the set of vertices without k. Then our all-pairs max-flow program on a graph $G = (V, E)$ can be written as

$$\begin{aligned}
&\text{Maximize} \quad \sum_{\substack{s,t\in V_{-k} \\ s\neq t}} v_{s,t} \\
&\text{subject to} \\
&\qquad \sum_{j:(i,j)\in E} x_{i,j,s,t} - \sum_{k:(k,i)\in E} x_{k,i,s,t} = \begin{cases} v_{s,t} & \text{if } i = s \\ -v_{s,t} & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \\
&\qquad \text{for all } i \in V \text{ for all } s,t \in V_{-k} \\
&\qquad x_{i,j,s,t} \leq 1 \text{ for all } (i,j) \in E \text{ for all } s,t \in V_{-k} \\
&\qquad v_{s,t} \geq 0 \text{ for all } s,t \in V_{-k} \\
&\qquad x_{i,j,s,t} \geq 0 \text{ for all } (i,j) \in E \text{ for all } s,t \in V_{-k}.
\end{aligned} \tag{3.3}$$

The optimal solution to this program has an objective function value equal to $Z_k(G)$.

Recall that the load of k is $\mathcal{L}_k(G) = Z_k(G) - Z_k(G \setminus \{k\})$. Therefore, we ultimately wish to choose subset S to maximize $Z_k(G \setminus S) - Z_k(G \setminus (S \cup \{k\})$.

If we have an optimization program maximizing an objective function $c^T x$, then it is minimizing $-c^T x$. This creates a problem because we cannot combine the two all-pairs max-flow problems for $Z_k(G)$ and $Z_k(G \setminus \{k\})$

into a single maximization program. Israeli and Wood (2002) present a solution to this problem. If we are maximizing the difference between two maximization problems, then we can take the dual of the second problem, which converts the problem into a minimization problem.

Given an optimization program,

$$\begin{array}{lrcl} \text{Maximize} & c^T x & & \\ \text{subject to} & & & \\ & Ax & \leq & b \\ & x & \geq & 0 \end{array} \tag{3.4}$$

the dual of Equation 3.4 is

$$\begin{array}{lrcl} \text{Minimize} & b^T y & & \\ \text{subject to} & & & \\ & A^T y & \geq & c \\ & y & \geq & 0. \end{array} \tag{3.5}$$

By the strong duality theorem, if an optimization problem has a finite optimal solution, the dual of that problem has the same finite optimal objective function value. Thus, we can use the dual of our program without changing the optimal objective function value.

Therefore, to compute $Z_k(G \setminus \{k\})$, we can take the dual of Equation 3.3 with regards to $G' = G \setminus \{k\} = (V', E')$. The dual is written as

$$\begin{array}{ll} \text{Minimize} & \displaystyle\sum_{\substack{s,t \in V' \\ s \neq t}} \sum_{(i,j) \in E'} y_{i,j,s,t} \\ \text{subject to} & \\ & y_{i,s,t} - y_{j,s,t} + y_{i,j,s,t} \geq 0 \text{ for all } (i,j) \in E' \text{ for all } s,t \in V' \\ & -y_{s,s,t} + y_{t,s,t} \geq 1 \text{ for all } s,t \in V' \\ & \\ & y_{i,j,st} \geq 0 \text{ for all } (i,j) \in E' \text{ for all } s,t \in V' \\ & y_{i,s,t} \text{ unrestricted for all } i,s,t \in V'. \end{array} \tag{3.6}$$

The dual of a max-flow problem is called a min-cut problem. We can interpret the dual variables $y_{i,s,t}$ as node potentials. For every edge (i,j), if $y_{i,s,t} < y_{j,s,t}$, or node i has lower potential than node j, then $y_{i,j,s,t} > 0$ and is counted in our objective function.

Thus, if we want to compute the load of k equal to $Z_k(G) - Z_k(G \setminus \{k\})$, we can use the following optimization program:

$$
\begin{aligned}
&\text{Maximize} \quad \sum_{\substack{s,t \in V_{-k} \\ s \neq t}} v_{s,t} - \sum_{\substack{s,t \in V' \\ s \neq t}} \sum_{(i,j) \in E'} y_{i,j,s,t} \\
&\text{subject to} \\
&\qquad \sum_{j:(i,j) \in E} x_{i,j,s,t} - \sum_{k:(k,i) \in E} x_{k,i,s,t} = \begin{cases} v_{s,t} & \text{if } i = s \\ -v_{s,t} & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \\
&\qquad \text{for all } i \in V \text{ for all } s,t \in V_{-k} \\
&\qquad x_{i,j,s,t} \leq 1 \text{ for all } (i,j) \in E \text{ for all } s,t \in V_{-k} \\
&\qquad y_{i,s,t} - y_{j,s,t} + y_{i,j,s,t} \geq 0 \text{ for all } (i,j) \in E' \text{ for all } s,t \in V' \\
&\qquad -y_{s,s,t} + y_{t,s,t} \geq 1 \text{ for all } s,t \in V' \\
&\qquad v_{s,t} \geq 0 \text{ for all } s,t \in V_{-k} \\
&\qquad x_{i,j,s,t} \geq 0 \text{ for all } (i,j) \in E \text{ for all } s,t \in V_{-k} \\
&\qquad y_{i,j,st} \geq 0 \text{ for all } (i,j) \in E' \text{ for all } s,t \in V' \\
&\qquad y_{i,s,t} \text{ unrestricted for all } i,s,t \in V'.
\end{aligned} \tag{3.7}
$$

Now that we have a linear program for the load of k in G, we can use this to find the load of k in $G \setminus S$. To do this, we will add some new variables to represent our subset and the effect removing the subset has on the graph.

3.2 Choosing a Subset

Let's return to LOMAX, where we want to find a subset of vertices that maximizes the load of k. If we let S represent the subset of vertices to remove from the graph, we want to maximize the objective function

$$\mathcal{L}_k(G \setminus S) = Z_k(G \setminus S) - Z_k(G \setminus (S \cup \{k\})). \tag{3.8}$$

As shown above, if we know $G \setminus S$, then we can use the optimization program in Equation 3.7 to solve for the load of k. Therefore, we want some method of altering G as we choose our subset that reflects our choice.

We can formalize this by creating variables z_i for each vertex i where $z_i = 0$ if z_i is in our subset and $z_i = 1$ if not. Consider the subproblem in Equation 3.3, which computes $Z_k(G)$. In our problem, the capacity of each present edge is one. Therefore, to account for our subset we create a variable $w_{i,j}$, representing the new capacity of edge (i,j) in Equation 3.3,

where

$$\begin{array}{rcl} w_{i,j} & \geq & 0 \\ w_{i,j} & \leq & z_i \\ w_{i,j} & \leq & z_j \\ w_{i,j} & \geq & z_i + z_j - 1. \end{array} \tag{3.9}$$

If the edge (i, j) is interrupted (meaning z_i or z_j is in the subset), then $w_{i,j}$ will be 0, and we cannot have any flow along that edge. If z_i and z_j are not in our subset, then $w_{i,j} \geq 1$ and $w_{i,j} \leq 1$. Thus, $w_{i,j} = 1$ and we can let $w_{i,j}$ be the capacity on the edge (i, j), essentially allowing us to delete edges as we add vertices to our subset. The constraint

$$x_{i,j,s,t} \leq 1$$

in Equation 3.7 is now written as

$$x_{i,j,s,t} \leq w_{i,j}.$$

For the subproblem in Equation 3.6 we want the $w_{i,j}$ to take on a slightly different meaning. As mentioned above, the dual variables can be interpreted as node potentials. If an edge (i, j) does not exist, or the nodes are at the same potential, then $y_{i,j,s,t} = 0$ and is not counted in our objective function.

Thus, we can substitute the constraint

$$y_{i,s,t} - y_{j,s,t} + y_{i,j,s,t} \geq 0$$

with a similar constraint that represents this relationship,

$$y_{i,s,t} - y_{j,s,t} + y_{i,j,s,t} \geq -M(1 - w_{i,j}).$$

M is an arbitrarily large constant. If z_i or z_j is in our subset, then $w_{i,j} = 0$, and $y_{i,j,s,t}$ can be set to zero regardless of the values of $y_{i,s,t}$ and $y_{j,s,t}$, and it essentially does not exist. Otherwise, if $w_{i,j} = 1$, the constraint reverts to the original.

Introducing the variables z_i and $w_{i,j}$ and the modifications on our load constraints, we can now write the full mixed-integer linear program. Given a graph $G = (V, E)$, a key vertex k, and a maximum number of vertices that can be removed m, the following mixed-integer linear program determines the maximum load of k.

The objective function is the load of k in a graph $G \setminus S$, and the optimal solution reveals which subset S gives this result:

$$\text{Maximize} \quad \sum_{\substack{s,t \in V_{-k} \\ s \neq t}} v_{s,t} - \sum_{\substack{s,t \in V' \\ s \neq t}} \sum_{(i,j) \in E'} y_{i,j,s,t}$$

subject to

$$\sum_{i \in V} z_i \geq n - m$$
$$z_k = 1$$
$$w_{i,j} \geq 0$$
$$w_{i,j} \leq z_i$$
$$w_{i,j} \leq z_j$$
$$w_{i,j} \geq z_i + z_j - 1$$

$$\sum_{j:(i,j) \in E} x_{i,j,s,t} - \sum_{k:(k,i) \in E} x_{k,i,s,t} = \begin{cases} v_{s,t} & \text{if } i = s \\ -v_{s,t} & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \quad \text{for all } i \in V \text{ for all } s,t \in V_{-k} \tag{3.10}$$

$$x_{i,j,s,t} \leq w_{i,j} \text{ for all } (i,j) \in E \text{ for all } s,t \in V_{-k}$$
$$y_{i,s,t} - y_{j,s,t} + y_{i,j,s,t} \geq -M(1 - w_{i,j}) \text{ for all } (i,j) \in E' \text{ for all } s,t \in V'$$

$$-y_{s,s,t} + y_{t,s,t} \geq 1 \text{ for all } s,t \in V'$$

z_i binary
$w_{i,j} \geq 0$
$v_{s,t} \geq 0$ for all $s,t \in V_{-k}$
$x_{i,j,s,t} \geq 0$ for all $(i,j) \in E$ for all $s,t \in V_{-k}$
$y_{i,j,st} \geq 0$ for all $(i,j) \in E'$ for all $s,t \in V'$
$y_{i,s,t}$ unrestricted for all $i,s,t \in V'$

3.3 Implementation

This formulation was translated into AMPL, a language for modeling optimization problems. Random graphs were created using an Erdős-Rényi graph generator in Python (in Erdős-Rényi graphs every possible edge occurs independently with probability p) that takes in a number of vertices and the probability p. A weighted random number generator is used to determine which edges exist.

Matrix	n	$\|E\|$	Run Time (s)	Subset	Max Load	Init Load
Matrix20a	20	48	6.75	$\{1,7,8,19\}$	62	38
Matrix20b	20	30	0.23	$\{1,2,7,11,18\}$	67	67
Matrix25a	25	64	45.35	$\{3,11,14,15\}$	101	23
Matrix25b	25	66	188.71	$\{8,14,21,22\}$	96	64
Matrix30a	30	68	65.63	$\{0,5,22,25\}$	153	112
Matrix30b	30	82	378.05	Ran out of memory		

Table 3.1 Run times and results of the MIP representing LOMAX on graphs with twenty to thirty vertices. The first column, Matrix, contains the filename for the graph in matrix format while the second and third columns contain the size of the vertex and edge set, respectively. Run Time (s) represents the total time taken to run the MIP in CPLEX and return a solution, while the optimal solution and objective function value are contained in the next two columns, Subset and Max Load. Lastly, the initial load of k in G is included in the column labeled Init Load for reference.

The initial graphs tested used a probability of 0.5 for each edge. These tests were done to find the best single vertex to remove on graphs of size 6, 7, 8, 9, 10, 12, 13, and 15 (three tests done on those with ≥ 10 vertices). The results matched those of the brute force algorithm. These same test graphs were also tested with $m = n - 1$ to find the best subset to remove, although often the best subset was a single vertex. The results were compared against those calculated by brute force (calculating the load of every possible subset).

For graphs with more than fifteen vertices, the mixed-integer program became more difficult to test since running it would often crash. One test on fifteen vertices and one on sixteen vertices returned results. These were matched against the output of the genetic algorithm. In the first case, the two had equal results. In the second case, the optimization program returned a subset with a greater load effect than the genetic algorithm.

When the probability of edges is reduced to 0.1 test results came back for graphs with up to thirty vertices, but no larger. All tests that returned results took less than four minutes. k was chosen randomly amongst vertices having at least two neighbors. Results are shown in Table 3.1.

More tests than those indicated were done but not recorded when testing the model. However, the run times were similar. Although the subsets contain four or more vertices, at least one vertex in each subset is not connected to k. One thing that is notable is that the maximum load significantly increases when these subsets are removed. This indicates that the

payoff for identifying a maximal subset is potentially large.

Matrix30b was the first graph with $p = 0.1$ to cause memory issues, and it was found that no other generated graph with thirty vertices generated results unless the resulting subset was empty. These generated graphs all had greater than 75 edges whereas Matrix30a, which returned an optimal subset, had only 68 edges, indicating that the number of edges influences the computational complexity.

Since the MIP framework appeared to be too computationally difficult to be practical, we needed to approach solving the model in a new way. In Chapter 4, we explain how the mixed-integer linear program was decomposed using Bender's decomposition. Additionally, Chapter 5 discusses properties of vertices in subsets with a positive load effect.

Chapter 4

Decomposition

Because the number of constraints grows on the order of $O(mn^2)$, where $n = |V|$ and $m = |E|$, we determined that we should use a decomposition algorithm to solve our problem for a given graph G. Bender's decomposition is an efficient decomposition algorithm for problems that have a small integer master problem and a large linear subproblem that is guaranteed to be feasible (Bertsimas and Tsitsiklis, 1997). In our case, the integer master problem is choosing the subset of vertices to remove. This problem has relatively few constraints. If the subset is fixed, we are left with a large linear network flow subproblem that is guaranteed to be integer and is relatively easy to solve by the simplex algorithm.

4.1 Bender's Decomposition

Bender's decomposition uses duality and delayed constraint generation to efficiently solve large mixed-integer linear programs. Consider a mixed-integer linear program

$$\begin{array}{lrcl} \text{Maximize} & \mathbf{c}^T\mathbf{x} + \mathbf{f}^T\mathbf{y} & & \\ \text{subject to} & & & \\ & A\mathbf{x} & = & \mathbf{b} \\ & B\mathbf{x} + D\mathbf{y} & = & \mathbf{d} \\ & \mathbf{x} \text{ int} & \geq & 0 \\ & \mathbf{y} & \geq & 0. \end{array} \tag{4.1}$$

Notice that the two variable vectors $\mathbf{x}$ and $\mathbf{y}$ exist in coupled constraints, constraints in which both $\mathbf{x}$ and $\mathbf{y}$ are present. That is, we cannot separate the constraints into those containing $\mathbf{x}$ and those not containing $\mathbf{x}$.

Given a fixed $\mathbf{x}^*$, the linear subproblem becomes

$$\begin{array}{lrcl} \text{Maximize} & \mathbf{f}^T\mathbf{y} & & \\ \text{subject to} & & & \\ & D\mathbf{y} & = & \mathbf{d} - B\mathbf{x}^* \\ & \mathbf{y} & \geq & 0 \end{array} \tag{4.2}$$

with dual

$$\begin{array}{lrcl} \text{Minimize} & (\mathbf{d} - B\mathbf{x}^*)^T\mathbf{p} & & \\ \text{subject to} & & & \\ & D^T\mathbf{p} & \leq & \mathbf{f} \\ & \mathbf{p} & & \text{unrestricted.} \end{array} \tag{4.3}$$

Suppose the dual minimizes over the polyhedron $P = \{\mathbf{p} | \mathbf{p}^T D \leq \mathbf{f}^T\}$, where P has I extreme points. Then we denote the extreme points $\mathbf{p}^i$ where $i = 1, 2, \ldots, I$. Furthermore, let $\mathbf{w}^j$ where $j = 1, 2, \ldots, J$ be the extreme rays of P. If the primal subproblem is feasible, by the strong duality theorem, the dual cannot be unbounded. Thus, for $\mathbf{x}$ to be feasible, we need $(\mathbf{w}^j)^T(\mathbf{d} - B\mathbf{x}) \geq 0$. Otherwise, an extreme ray would be an improving direction, and our dual would be unbounded.

Furthermore, the optimal objective function of the subproblem has to be greater than or equal to the objective function at all extreme points. In other words, $(\mathbf{p}^i)^T(\mathbf{d} - B\mathbf{x}) \geq z$ for all i where z represents the optimal objective function of the primal subproblem.

Therefore, we can reformulate the master problem in Equation 4.1 to be

$$\begin{array}{lrcl} \text{Maximize} & \mathbf{c}^T\mathbf{x} + z & & \\ \text{subject to} & & & \\ & A\mathbf{x} & = & \mathbf{b} \\ & (\mathbf{p}^i)^T(\mathbf{d} - B\mathbf{x}) & \geq & z \text{ for all } i \\ & (\mathbf{w}^j)^T(\mathbf{d} - B\mathbf{x}) & \geq & 0 \text{ for all } j \\ & \mathbf{x} \text{ int} & \geq & 0 \\ & z & \geq & 0. \end{array} \tag{4.4}$$

Although we have reduced the number of variables, the number of constraints is extremely large. Therefore, we can use delayed constraint generation to improve the efficiency of this algorithm.

4.2 Delayed Column Generation

In delayed constraint generation the master problem in Equation 4.4 begins in the form

$$\begin{array}{lrcl} \text{Maximize} & \mathbf{c}^T\mathbf{x} + z & & \\ \text{subject to} & & & \\ & A\mathbf{x} & = & \mathbf{b} \\ & \mathbf{x} \text{ int} & \geq & 0 \\ & z & \geq & 0 \end{array} \tag{4.5}$$

with no upper bound on what z can be.

Given a feasible solution to this problem $\mathbf{x}^*, z^*$ we then solve the dual of the linear subproblem Equation 4.3 with $\mathbf{x}^*$ substituted in. If the dual is unbounded, then we have identified an extreme ray $\mathbf{w}^j$ whose constraint is violated. Therefore, we add in the constraint

$$(\mathbf{w}^j)(\mathbf{d} - B\mathbf{x}) \geq 0.$$

If the dual returns a finite optimal solution $\mathbf{p}^i$ with objective function less than z^*, then we must add in the constraint

$$(\mathbf{p}^i)(\mathbf{d} - B\mathbf{x}) \geq z.$$

Otherwise all constraints have been satisfied, and we have reached the optimal solution.

4.3 Application

We now apply Bender's Decomposition to our mixed-integer linear program in Equation 3.10. In this case we have coupled constraints with our $w_{i,j}$, $x_{i,j,s,t}$, $y_{i,j,s,t}$ and $y_{i,s,t}$ variables. We let the z_i's and $w_{i,j}$'s be the variables in our master problem. Our initial master problem contains only the constraints related to the $w_{i,j}$'s and z_i's, representing the choice of subset to

remove. Thus the master problem is

$$\begin{array}{ll} \text{Maximize} & \mathcal{L}_k \\ \text{subject to} & \\ & \sum_{i \in V} z_i \geq n - m \\ & z_k = 1 \\ & w_{i,j} \geq 0 \\ & w_{i,j} \leq z_i \\ & w_{i,j} \leq z_j \\ & w_{i,j} \geq z_i + z_j - 1 \\ & \\ & z_i \text{ binary} \\ & w_{i,j} \geq 0 \\ & \mathcal{L}_k \geq 0. \end{array} \tag{4.6}$$

Here, $\mathcal{L}_k$ represents the optimal load of k. It currently has no restrictions on its value.

Solving Equation 4.6 determines a feasible $\mathbf{z}$ and $\mathbf{w}$, which we can use to compute the load of k in the dual of the linear subproblem. When taking the dual we let $x'_{i,s,t}$ be the dual variables corresponding to the flow balance constraints of the $x_{i,j,s,t}$'s and $x'_{i,j,s,t}$ be the dual variables corresponding to the capacity constraints on the $x_{i,j,s,t}$'s. Similarly, we let $y'_{i,j,s,t}$ be the dual variables corresponding to the edge constraints on $y_{i,j,s,t}$ and $y'_{i,j,s,t}$ be the dual variables corresponding to the constraints on the relationship between $y_{s,s,t}$ and $y_{t,s,t}$. The linear subproblem becomes

$$\begin{array}{ll} \text{Minimize} & \displaystyle\sum_{\substack{s,t \in V_{-k} \\ s \neq t}} \sum_{(i,j) \in E} w_{i,j} x'_{i,j,s,t} + \sum_{\substack{s,t \in V' \\ s \neq t}} y'_{s,t} - \sum_{\substack{s,t \in V' \\ s \neq t}} \sum_{(i,j) \in E'} M(1 - w_{i,j}) y'_{i,j,s,t} \\ \text{subject to} & \\ & x'_{i,s,t} - x'_{j,s,t} + x'_{i,j,s,t} \geq 0 \text{ for all } (i,j) \in E \text{ for all } s,t \in V_{-k} \\ & -x'_{s,s,t} + x'_{t,s,t} \geq 1 \text{ for all } s,t \in V_{-k} \\ & \displaystyle\sum_{j:(i,j) \in E'} y'_{i,j,s,t} - \sum_{k:(k,i) \in E'} y'_{k,i,s,t} = \begin{cases} y'_{s,t} & \text{if } i = s \\ -y'_{s,t} & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \end{array}$$

(cont.)

$$
\begin{aligned}
&\text{for all } i, s, t \in V' \\
&y'_{i,j,s,t} \geq -1 \text{ for all } (i,j) \in E' \text{ for all } s, t \in V' \\
&x'_{i,j,s,t} \geq 0 \text{ for all } (i,j) \in E \text{ for all } s, t \in V_{-k} \\
&x'_{i,s,t} \text{ unrestricted for all } i, s, t \in V_{-k} \\
&y'_{s,t} \leq 0 \text{ for all } s, t \in V' \\
&y'_{i,j,s,t} \leq 0 \text{ for all } (i,j) \in E' \text{ for all } s, t \in V'.
\end{aligned} \tag{4.7}
$$

An AMPL script file was written to perform Bender's decomposition on the problem for a given graph. At the beginning of each iteration c, the master is solved and we obtain the optimal z_i's and $w_{i,j}$'s. Initially, we start with an infinite objective function and all $z_i = 1$. The dual of the linear subproblem, shown in Equation 4.7, is then solved with the $w_{i,j}$'s substituted in.

If the subproblem is unbounded, AMPL returns the extreme ray, defining $\mathbf{x}'_c$ and $\mathbf{y}'_c$, and we add in the constraint

$$\sum_{\substack{s,t \in V_{-k} \\ s \neq t}} \sum_{(i,j) \in E} w_{i,j} x'_{i,j,s,t,c} + \sum_{\substack{s,t \in V' \\ s \neq t}} y'_{s,t,c} - \sum_{\substack{s,t \in V' \\ s \neq t}} \sum_{(i,j) \in E'} M(1 - w_{i,j}) y'_{i,j,s,t,c} \geq 0.$$

If the subproblem has an objective function value less than or equal to the current MaxFlowEffect, then we add in the constraint

$$\sum_{\substack{s,t \in V_{-k} \\ s \neq t}} \sum_{(i,j) \in E} w_{i,j} x'_{i,j,s,t,c} + \sum_{\substack{s,t \in V' \\ s \neq t}} y'_{s,t,c} - \sum_{\substack{s,t \in V' \\ s \neq t}} \sum_{(i,j) \in E'} M(1 - w_{i,j}) y'_{i,j,s,t,c} \geq \mathcal{L}_k.$$

Otherwise, the algorithm terminates.

4.4 Implementation

The decomposition algorithm was fully implemented and tested for small graphs of fifteen or fewer vertices and returned the same optimal subsets as the MIP. The algorithm was then tested on graphs having between twenty and thirty-five vertices. It was found that when the algorithm encountered a graph that contained no vertex subsets with a positive load effect, the

algorithm terminated within a second. However, when identifying a subset having positive load effect, the algorithm would run out of memory after completing around 4000 iterations of Simplex. The total time the algorithm ran before running out of memory varied but was approximately three days.

After performing simplex on the subproblem (which is quite large) we store the values of variables to create the next cut. With each iteration, the amount of memory used increased linearly, which indicates that this process is what causes the memory to run out.

Research into different methods for fixing this problem did not yield any solutions. Therefore, Bender's Decomposition does not yield a practical method for solving our given model. This gives motivation for identifying properties of vertices in subsets with a positive load effect. Identifying such properties helps to eliminate potential subsets from consideration and reduces the complexity of the problem. This is discussed in Chapter 5.

Chapter 5

Structural Properties

Given that our model and decomposition algorithm for LOMAX proved impractical for graphs with more than thirty nodes, there remains work to be done to understand which subset to remove. Understanding the structural properties of vertices within subsets with a positive load effect on k can help give us insight. In particular, identifying necessary conditions for these vertices allows us to eliminate candidates and reduce the complexity of the LOMAX problem. This could be useful both for our model and future heuristics.

5.1 Previous Results on Single-LOMAX

Martonosi et al. (2011) prove four theorems about single vertex subsets that cannot improve the load of k. These are presented below and will prove useful when looking at larger subsets.

Theorem 5.1. *(Martonosi et al., 2011) Given a graph G with a key vertex k of degree two, removing any vertex i adjacent to vertex k will not increase the load of k.*

Theorem 5.2. *(Martonosi et al., 2011) Let G be a chordless n-cycle. Then for any choice of key vertex k in G, the load of k cannot by increased by removing another vertex.*

Theorem 5.3. *(Martonosi et al., 2011) Let k and i be distinct vertices in graph G. If there is only one edge-disjoint path between k and i, then* $\mathcal{E}_k(G, \{i\}) \leq 0$.

Theorem 5.4. *(Martonosi et al., 2011) Let k and i be distinct vertices in graph G. Consider an edge cut C that partitions G into two components such that i and k*

are in separate components. Let G_k be the subgraph of G over the set of vertices in the component containing k, and let G_i be the subgraph of G over the set of vertices in the component containing i. Let $i_1, \ldots, i_p$, where $p \leq |C|$, be the set of vertices on the i side of the cut that are adjacent to G_k. Let $k_1, \ldots, k_s$, where $s \leq |C|$, be the set of vertices on the k side of the cut adjacent to G_i. Suppose any boundary vertex $i_1 \in G_i$ has at least $\lfloor |C|/2 \rfloor$ edge-disjoint paths to every other boundary vertex in G_i by using only vertices in $G_i \setminus \{i\}$, and any boundary vertex $k_1 \in G_i$ has at least $\lfloor |C|/2 \rfloor$ edge-disjoint paths to every other boundary vertex in G_k by using only vertices in $G_k \setminus \{k\}$. Then the load effect on k by removing vertex i is nonpositive.

5.2 LOMAX

We first work to extend these results for any size subset. Theorem 5.1 and Theorem 5.3 can be extended naturally. (Note that Theorem 5.2 already eliminates any size subset from consideration.) We also add in a theorem about vertices that are not in a cycle with k.

Theorem 5.5. *Given a graph G with key vertex k having degree two, any vertex subset containing i adjacent to vertex k will not increase the load of k.*

Proof. By removing vertex i adjacent to k, k becomes a leaf and has load 0, regardless of what other vertices we choose for our subset. Since load is always nonnegative, a load of k of 0 after removing i can be no greater than the load of k in the original graph. □

Theorem 5.6. *Let G be a graph with key vertex k and let i be a vertex with a single edge-disjoint path to k. Let S be any vertex subset containing i, and let $T = S \setminus \{i\}$. Then, $\mathcal{L}_k(G \setminus S) \leq \mathcal{L}_k(G \setminus T)$. Therefore, any subset T that excludes i will have at least as large a load effect on k as $T \cup \{i\}$.*

Proof. Let G be a graph with key vertex k and let i be a vertex with a single edge-disjoint path to k. Consider a subset S containing i. Let $T = S \setminus \{i\}$. We first note that in $G \setminus T$, i can have at most one edge-disjoint path to k. Otherwise, an alternate path would have existed in G, contradicting our statement about i. Suppose i does not have a path to k, then i has no effect on the load of k when we add it to T, and $\mathcal{L}_k(G \setminus S) = \mathcal{L}_k(G \setminus T)$.

Otherwise, i has exactly one edge-disjoint path to k. By Theorem 5.3, i cannot have a positive load effect on k in $G \setminus T$. Thus, $\mathcal{L}_k(G \setminus S) \leq \mathcal{L}_k(G \setminus T)$. Any subset will have a greater or equal load effect on k without i, as desired. □

Theorem 5.7. *Let G be a graph with key vertex k and let i be a vertex with a single vertex-disjoint path to k. Let S be any vertex subset containing i, and let $T = S \setminus \{i\}$. Then, $\mathcal{L}_k(G \setminus S) \leq \mathcal{L}_k(G \setminus T)$. Therefore, any subset T that excludes i will have at least as large a load effect on k as $T \cup \{i\}$.*

Proof. Let G be a graph with key vertex k and let i be a vertex with a single vertex-disjoint path to k. Consider a subset S containing i. Let $T = S \setminus \{i\}$. We first note that in $G \setminus T$, i either has a single edge-disjoint path to k or more than one edge-disjoint path to k but a single vertex-disjoint path to k.

If i has a single edge-disjoint path, by Theorem 5.6, $\mathcal{L}_k(G \setminus S) \leq \mathcal{L}_k(G \setminus T)$.

Otherwise, i has more than one edge-disjoint path to k but a single vertex-disjoint path to k. Since i only has a single vertex-disjoint path to k, it is not in a cycle with k. Thus, there is some vertex v through which every edge-disjoint path between i and k must pass. Let V_i be the vertices in i's component in $G \setminus \{v\}$ and let V_k be the vertices in k's component of $G \setminus \{v\}$.

The paths between vertices in v_i do not contribute to the load of k. This is because any edge-disjoint path that contains k must cross back from V_k into V_i through v and we can restrict ourselves to paths that only use v and vertices in V_i.

Thus, when i is removed, we need to look at its effect on the edge-disjoint paths between vertices in V_i and V_k. Without i, the flow in V_i is affected. However, once a path reaches v, it can take the same path as previously. Thus, we cannot possibly force more edge-disjoint paths through k and $\mathcal{L}_k(G \setminus S) \leq \mathcal{L}_k(G \setminus T)$, as desired.

□

These theorems show that we can ignore vertices adjacent to k if k has degree two and ignore all vertices with a single edge-disjoint path to k or have no cycle with k. Based on these results we can perform some simplifications to our graph.

5.2.1 Simplifying the Graph

Since vertices with one edge-disjoint path to k cannot increase the load effect on k of any subset, we can simplify the graph. For each vertex u with one edge-disjoint path to k, if its neighbor v also has one edge-disjoint path to k, we can combine these vertices into a single vertex. To do this we let v remain in the graph. If u has a neighbor w that is not adjacent to v, then

we add an edge between v and w. Thus, any vertex adjacent to u or v originally will be adjacent to v. An example of this is the combination of vertices 3 and 4 in Figure 5.1.

Additionally, if we have a vertex v with more than one edge-disjoint path to k and with at least two neighbors $u_1, u_2, \ldots, u_s$ that a have a single edge-disjoint path to k, we can combine v's neighbors into a single vertex u. This is different from the case above. In the first case, we were combining vertices whose single edge-disjoint path must pass through a common edge adjacent to v. Now, we are combining vertices whose single edge-disjoint path must pass through v. An example of this is the combination of vertices 8, 9, and 10 in Figure 5.1.

To account for this simplification process, we need to associate a weight function to each vertex. Every vertex starts with a weight of one. Whenever we combine vertices we add the weights of those vertices together to get the weight of the new combined vertex. When calculating the number of edge-disjoint paths between any pair of vertices u and v in the graph, we multiply the number of paths by the product of the weights of the vertices.

If both u and v have more than one edge-disjoint path to k, this does not affect anything. For example, in Figure 5.1, the number of edge-disjoint paths between vertex 2 and vertex 6 is multiplied by one. Otherwise, if v is a combined vertex then there is a single edge-disjoint path between the pair (since we only combine vertices with a single edge-disjoint path to k and any combined vertex has a single neighbor). Furthermore, u could only have one edge-disjoint path to any vertex combined into v so multiplying by the weight accounts for these extra paths. For example, in Figure 5.1, the number of edge-disjoint paths between vertex 2 and vertex 8 is multiplied by three, representing the single edge-disjoint paths vertex 2 had to vertices 8, 9, and 10.

The whole simplification process is given below.

Simplification Process

- Combine any vertex with a single edge-disjoint path to k

Figure 5.1 shows an example of this simplification process. Note that vertices 8, 9, and 10 have been combined together into a vertex with weight three. Additionally, vertices 3 and 4 have been combined together with weight two. At the end of this process every vertex with more than one edge-disjoint path to k will have at most one neighbor with a single edge-disjoint path to k.

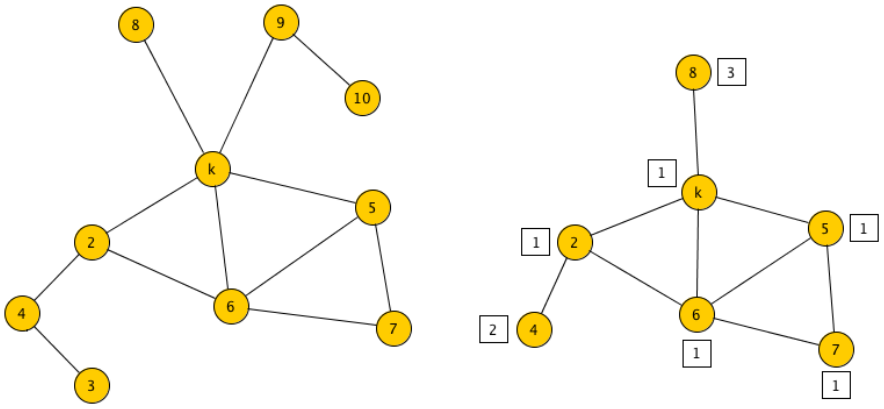


Figure 5.1 An example of a graph (left) and its simplified version (right) with vertex weights. Vertices 8, 9, and 10 have been combined together into vertex with weight three. Additionally, vertices 3 and 4 have been combined together into vertex 4 with weight two.

As argued above, the number of edge-disjoint paths between vertices that have not been combined together are still counted when we multiply the weight functions of s and t. For example, in Figure 5.1, we count 6 paths between vertex 8 and vertex 4, accounting for all the paths between vertices 3 and 4 and vertices 8, 9, and 10.

However, some paths between vertices that have been combined together are no longer counted. For example, the path between vertex 8 and 9 in Figure 5.1 is no longer counted in the load of k. Luckily, we never want to remove a vertex v that disconnects any of our combined vertices from k since v's removal would only decrease the load of k. Therefore, these edge-disjoint paths between vertices will still exist when we remove an optimal subset and their effect on the load of k is a constant. Thus, our optimization problem still finds an optimal subset to remove on the simplified graph that is optimal in the original graph.

5.3 Examining Cyclic Structures

Removing a vertex affects the load on a key vertex when it disconnects some cycles, changing where flow can travel. Therefore, we are primarily interested in how the cyclic structure of the graph changes when we remove our subset. Below, we analyze some simple cyclic structures and de-

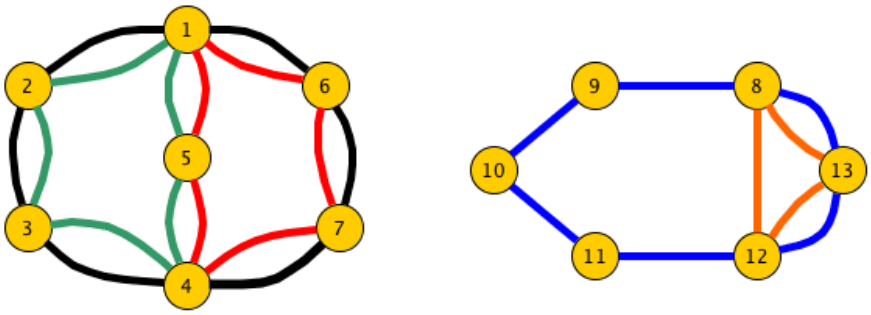


Figure 5.2 Examples of chordless cycles. On the left, Cycle 1 (red), 2 (green), and 3 (black) are all chordless cycles. The black cycle does not contain either the red or green cycle because it does not contain vertex 5. On the right, Cycle 4 (blue) is an example of a cycle that is not chordless since it contains Cycle 5 (orange).

termine which vertices we should consider removing. These demonstrate how knowing information about the cycles k is in can make calculating the load effect of vertices much easier.

A cycle can be represented as the vertices within that cycle. In this section, we define a *chordless cycle* to be a cycle that does not contain a smaller cycle. For an example, see the black, green, red, and orange cycles in Figure 5.2. We consider a vertex v to be in a *single chordless cycle* if v has two neighbors u and w such that any chordless cycle containing v contains u and w. For example, vertex 2 in Figure 5.2 is in a single chordless cycle, but vertex 1 is not.

In order to determine if we should remove a vertex we look at a couple of specific cases. These cases will be defined based on the cyclic structure of the graph. First, we analyze a graph with a single cycle. Next, we examine cycles in parallel (defined in Section 5.3.2) and when k is in a single chordless cycle (defined above). Restricting ourselves to these specific cases allows us to determine which vertices we should include in our subset.

Let $D(v)$ be the number of vertices disconnected from k when vertex v is removed. We also define a metric $E(v)$. If v has an adjacent vertex u with one edge-disjoint path to k, then $E(v)$ is the weight of u. Otherwise, $E(v) = 0$. Similarly, $E(S)$, where $S = \{s_1, s_2, \ldots, s_r\}$ is a subset of vertices, is $E(s_1) + E(s_2) + \cdots + E(s_r)$.

Note that $E(k)$ is the number of vertices with a single edge-disjoint path

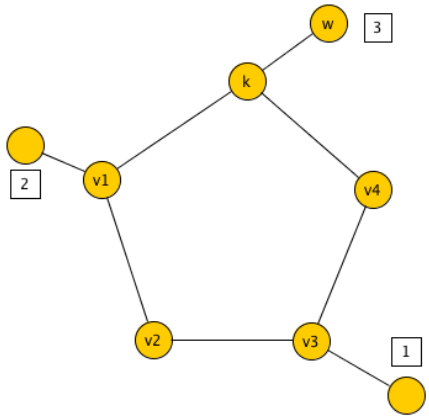


Figure 5.3 An example of a graph G that has a single cycle of length $l+1=5$ containing k. Here $E(k)=3$, $E(v1)=2$, and $E(v3)=1$.

to k such that the path does not contain any vertices with more than one edge-disjoint path to k. Since we assumed that the graph is simplified, if $E(k)>0$, there is exactly one vertex w adjacent to k with weight $E(k)$ (we will use w to denote this vertex throughout this section). For simplicity, we will assume w exists (if w does not, then $E(k)=0$ and it will not affect our calculations). This value is important because w has a single edge-disjoint path to every other vertex in the graph that must pass through k and contributes to the load of k.

5.3.1 Single Cycles

Theorem 5.8. *Suppose a graph $G=(V,E)$ has a single cycle of length $l+1$ containing k. Let $n=|V|$. We label the vertices around the cycle $k, v_1, v_2 \ldots, v_l$. Furthermore, let $I=\{v_1, v_2, \ldots, v_I\}$. Then, by definition, $E(I)=E(v_1)+E(v_2)+\cdots+E(v_i)$. Then, the load effect of vertex v_i is*

$$-E(k)(D(v_i))+(E(I-1)+i-1)(l-i+E(L)-E(I))-\binom{l}{2}.$$

See Figure 5.3 for an example where $l=4$, $E(k)=3$, $E(v_1)=2$, $E(v_3)=1$, and $E(I)=3$.

Proof. In this graph, each pair of vertices in our cycle has 2 edge-disjoint paths between them. After k is removed, these pairs have a single path be-

tween them. Thus, these pairs contribute $\binom{l}{2}$ to the load of k. Additionally, any vertex with a single edge-disjoint path to k adjacent to some vertex v_i in our cycle (of which there are $E(k)$) has exactly one path to any other vertex in the cycle or adjacent to some v_j regardless of whether k exists in the graph. Therefore, these pairs do not contribute to the load of k.

Lastly, w has a single edge-disjoint path to any other vertex that must pass through k. Let $m = n - 2$, which is the number of vertices in our cycle or adjacent to some v_i on the cycle (we subtract off k and w). Therefore, the initial load of k is

$$E(k)m + \binom{l}{2}.$$

For example, in Figure 5.3, $E(k) = 3$, $m = 6$, and $l = 4$. Thus, the load of k is $3 \cdot 7 + \binom{4}{2} = 33$.

Consider a vertex v_i. After we remove vertex v_i, all vertices in the graph have a single edge-disjoint path between them. Furthermore, $D(v_i)$ vertices are now disconnected from k. Therefore, we have $E(k)(m - D(v_i))$ paths from w to all other vertices that contribute to the load of k. Additionally, we have single edge-disjoint paths through k for each pair of vertices on either component off k formed when we remove v_i. There are

$$E(I-1) + i - 1$$

vertices in one component off k and $l - i + E(L) - E(I)$ vertices in the other component. This contributes to

$$(E(I-1) + i - 1)(l - i + E(L) - E(I))$$

paths. Thus, the load of k is now

$$(E(I-1) + i)(l - i + E(L) - E(I)) + E(k)(m - D(v_i))$$

For example, in Figure 5.3, if we were to remove v_2, We would have $i - 1 = 1$ vertices previously on the cycle plus $E(i-1) = E(\{v_1\}) = 2$ vertices that are adjacent to a v_i on one component off k, and we have $l - i = 2$ vertices previously on the cycle plus

$$E(L) - E(I) = E(\{v_1, v_2, v_3, v_4\}) - E(\{v_1, v_2\}) = 3 - 2 = 1$$

vertices that are adjacent to a v_i on the other component off k. Furthermore, $m - D(v_2) = 7 - 1 = 6$ and $E(k) = 3$. Thus, the load of k when we remove v_2 is $2 \cdot 3 + 6 \cdot 3 = 24$.

Thus, the load effect of v_i is

$$-E(k)(D(v_i)) + (E(I-1)+i-1)(l-i+E(L)-E(I)) - \binom{l}{2}.$$

□

We only want to remove v_i if its load effect on k is positive. Therefore, we need

$$-E(k)(D(v_i)) + (E(I-1)+i-1)(l-i+E(L)-E(I)) - \binom{l}{2} > 0.$$

Furthermore, since we never want to remove vertices with a single edge-disjoint path to k and removing any vertex in the cycle breaks all cycles in the graph, any subset of vertices with positive load effect on k has size 1. Therefore, if the above value is ≤ 0, we can mark v_i as ignorable.

5.3.2 Parallel Cycles

Theorem 5.9. *Suppose a graph G has p vertex-disjoint paths from k to a vertex u (forming $\binom{p}{2}$* parallel cycles*). Suppose all vertices not in these paths have a single path to k. We label the vertices along the i^{th} path of length l_i to be $v_{i,1}, v_{i,2}, \ldots, v_{i,l_i}$. Let $m = n - E(k) - 1$ and let m_c be the number of vertices in the p paths between k and u. Then, the load effect of $v_{i,j}$ is*

$$-E(k)D(v_{i,j}) - \frac{1}{2}(l_i^2 + l_i - 2m_c l_i) + (j - 1 + E(P_{i,j})(m - E(P_{i,j-1}) - j + 1),$$

and the load effect of u is

$$(\sum_{i<j}(E(P_i)+l_i)(E(P_j)+l_j)) + E(k)(m-1-E(u)) - E(k)m - \binom{m_c}{2}.$$

An example is shown in Figure 5.4 where $p = 3$.

Proof. Let $P_i = \{v_{i,1}, v_{i,2}, \ldots, v_{i,l_i}\}$. Similarly, we let $P_{i,j} = \{v_{i,1}, v_{i,2}, \ldots, v_{i,j}\}$ ($P_{i,j}$ is the set of the first j vertices in P_i).

Every pair of vertices on our parallel paths has two edge-disjoint paths between them. After k is removed, these pairs have a single path between them. Any vertex with a single edge-disjoint path to k that is attached to a $v_{i,j}$ has exactly one path to any other $v_{i,j}$ or vertex adjacent to a $v_{i,j}$ regardless

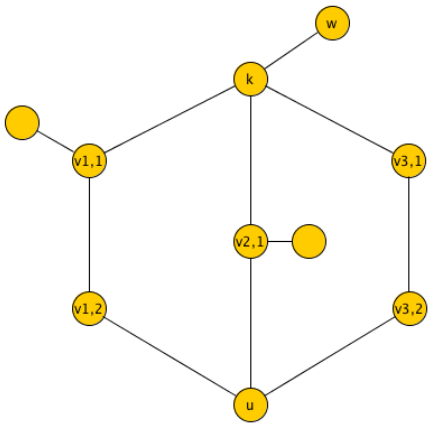


Figure 5.4 An example of a graph G that has $p = 3$ parallel paths from k to a vertex u.

of whether k exists. Lastly, w has one edge-disjoint path to every vertex in the cycle that must go through k. Therefore, the initial load of k is

$$E(k)m + \binom{m_c}{2}.$$

Now let $v_{i,j}$ be a vertex along path i (not equal to u or k). When we remove this vertex, one of our paths becomes broken. The number of paths from the $E(k)$ vertices off of k decreases by the number of vertices disconnected, equal to $D(v_{i,j})$. Furthermore, while we maintain $\binom{m_c - l_i}{2}$ paths through k for our vertices still in the cyclic structure, we now must add single paths between the newly disconnected part and our cycle. This contributes $(j-1+E(P_{i,j-1}))(m - E(P_{i,j-1}) - j + 1)$ paths. Thus, the new load of k is

$$E(k)(m - D(v_{i,j})) + (j-1+E(i,j-1))(m - E(i,j-1) - j + 1) + \binom{m_c - l_i}{2}.$$

Thus, the load effect of vertex $v_{i,j}$ is

$$-E(k)D(v_{i,j}) - \frac{1}{2}(l_i^2 + l_i - 2m_c l_i) + (j - 1 + E(P_{i,j})(m - E(P_{i,j-1}) - j + 1).$$

Now we consider the load effect of u. When we remove u, all paths become disconnected and we have a cycle free graph (we stated that the

only cycles were those created by the P_i's). Therefore, the load of k is given by

$$(\sum_{i<j}(E(P_i)+l_i)(E(P_j)+l_j))+E(k)(m-1-E(u)).$$

and the load effect of u is

$$(\sum_{i<j}(E(P_i)+l_i)(E(P_j)+l_j))+E(k)(m-1-E(u))-E(k)m-\binom{m_c}{2}.$$

Thus, we only consider removing u if it has a positive load effect or

$$(\sum_{i<j}(E(P_i)+l_i)(E(P_j)+l_j))+E(k)(m-1-E(u))>E(k)m+\binom{m_c}{2}.$$

□

The load effect of vertex $v_{i,j}$ can be calculated relatively simply by running through each path and tallying each $E(P_{i,j})$ since we simply check the weight of any vertex adjacent to $v_{i,j}$ with a single edge-disjoint path to k. This takes $O(n)$ amount of time after having found all-pairs max flow since we visit each vertex at most once. Then, for each vertex we can compute the load effect directly.

Suppose that this value is ≤ 0. Notice that if we remove some subset of vertices, $v_{i,j}$ either has a single edge-disjoint path to k, in which case we don't want to remove it, or the number of paths between k and u has decreased.

If $v_{i,j}$ still has at least two paths to k, then we can simplify the graph by the process in Section 5.2.1 and recompute the load effect of $v_{i,j}$. Whenever we remove a subset, we are breaking some paths between u and k. Thus, $E(k)$ increases, and m and m_c decrease. Furthermore, since we never consider removing vertices with a single edge-disjoint path to k, every $E(v_{i,j})$ remains constant. Thus, the new load effect decreases. It is still not beneficial to remove $v_{i,j}$.

If we compute the above value for some graph G, any vertex with a negative load effect on k in this case should be marked as ignorable since it is never beneficial to include this vertex in our subset.

5.3.3 k in a Single Chordless Cycle

Theorem 5.10. *Suppose k is in a single chordless cycle C of length $l+1$ but is in more than one cycle overall. Suppose that C has only two vertices u and v adjacent*

to three vertices in a cycle with k. Let l_k be the number of vertices on the path between u and v that contains k, not including k. Let m be the number of vertices in a cycle with k but not in C. Then,

- *The load of a vertex not in C is nonpositive.*
- *If s is a vertex on the path between u and v in C that does not contain k (not including u and v) whose removal splits the previous cyclic structure into two parts: one with n_1 vertices and one with n_2 vertices, then the load effect of s is at least*

$$E(k)(n - D(s) - E(k) - 1) + \binom{l'}{2} + l'_k(m + l' - l).$$

- *If s is a vertex on the path between u and v in C that contains k (including u and v) whose removal results in a new single chordless cycle C' of length l', the load effect of s is*

$$-E(k)D(s) + n_1 n_2 - \binom{l}{2} - l_k m.$$

For an example, see Figure 5.5. In this graph, C, consisting of the bolded edges, is our single chordless cycle of length $l + 1 = 6$. There are $l_k = 2$ vertices (a and b) on the path between u and v that contains k, not including k. Furthermore, there are $m = 5$ vertices in a cycle with k that are not in C.

Proof. There are $E(k)(n - E(k) - 1)$ paths between w and all other vertices in the graph. There are $\binom{l}{2}$ paths that must go through k since any pair of vertices on C has at least two edge-disjoint paths, one of which must go through k.

Furthermore, any vertex s in a cycle with k but not in C has at least two edge-disjoint paths to any vertex on the path between u and v in C that contains k, one of which must go through k. However, s has two edge-disjoint paths to any vertex on the path between u and v in C that does not contain k (not including u and v) neither of which must pass through k since we can just choose a direct path from u and a direct path from v.

Consider a pair (s, t) where s is any vertex not in C. Then if any edge-disjoint path between s and t passes through k it must not be using the other arc of C since u and v both have degree three (not considering edges to leaves). We can use this other path when k is removed without affecting the number of edge-disjoint paths between s and t. For example, in Figure 5.6,

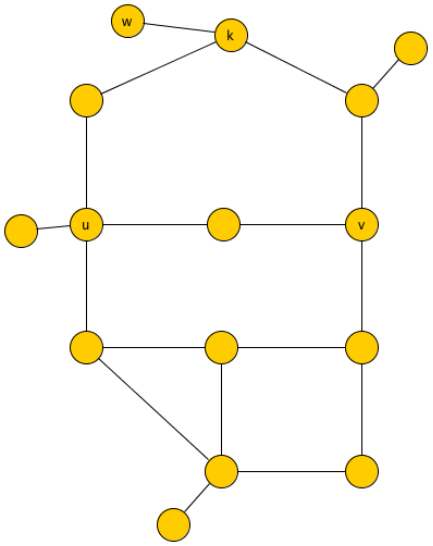


Figure 5.5 An example of a graph G where k is in a single chordless cycle C (whose edges are bolded) of length $l + 1 = 6$ but is in more than one cycle overall and C has only two vertices u and v adjacent to strictly greater than two vertices in a cycle with k.

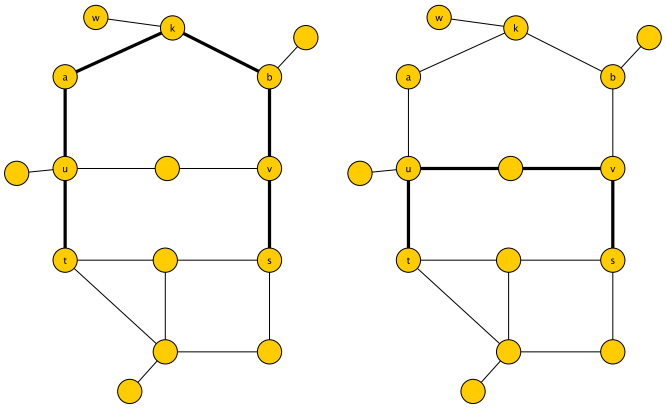


Figure 5.6 An example in which any edge-disjoint path between s and t not in C does not have to contain k. The path on the left passes through k, but the path on the right does not. Since u and v have degree three, we cannot have both paths.

we can either chose the path on the left through k or the path on the right that does not pass through k. Since u and v have degree three, we cannot have both paths.

Thus, the initial load of k is

$$E(k)(n - E(k) - 1) + \binom{l}{2} + l_k m. \tag{5.1}$$

Consider any vertex s in a cycle with k that is not C. Then, this cycle must contain u and v since u and v are the only vertices on C adjacent to strictly greater than two vertices in a cycle with k. Removing s maintains the general structure described above, but the number of vertices in the graph decreases. Therefore, the load effect of s on k is nonpositive.

Let s be a vertex on the path between u and v in C that does not contain k. If we remove s, k is in another single chordless cycle C' with length l'. Thus, the load effect on k is at least

$$E(k)(n - D(s) - E(k) - 1) + \binom{l'}{2} + l'_k(m + l' - l). \tag{5.2}$$

If Equation 5.2 $>$ Equation 5.1, then removing s has a positive load effect on k. If $G - s$ has a similar structure to G (the new single chordless cycle has only two vertices u' and v' adjacent to greater than two vertices in a cycle with k both with degree three), then we are guaranteed for the load effect on k to be exactly

$$E(k)(n - D(s) - E(k) - 1) + \binom{l'}{2} + l'_k(m + l' - l).$$

Thus, if the value decreases, s does not have a positive load effect on k. An example in which this occurs is when we have a stack of cycles. See Figure 5.7.

Let s be a vertex on the path between u and v that contains k (including u and v). If we remove s then all vertices have a single path to k. Suppose that this splits the previous cyclic structure into two parts: one with n_1 vertices and one with n_2 vertices ($n = n_1 + n_2 + E(k) + D(s) + 1$). Then, the load of k is

$$E(k)(n_1 + n_2) + n_1 n_2.$$

The difference in loads is

$$-E(k)D(s) + n_1 n_2 - \binom{l}{2} - l_k m.$$

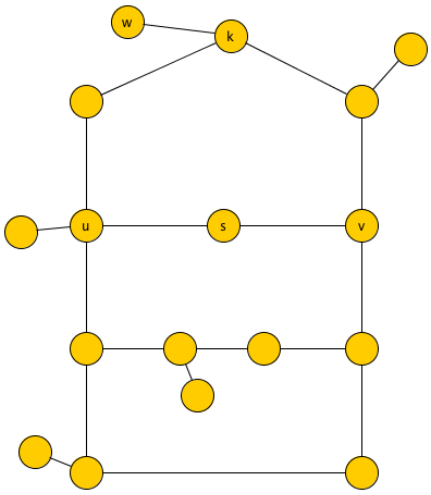


Figure 5.7 An example of a stack of cycles. When we remove s, we maintain the same general structure.

Note that if we remove a subset of vertices but maintain the general structure of G, then this number only decreases. In that case if the difference in loads is nonpositive we can mark s as ignorable. □

These cyclic structures appear quite frequently in smaller graphs and allow us to eliminate vertices from consideration. However, for larger graphs these cases rarely occur since the average vertex degree increases. Future work needs to be done to relate these cases together and generalize the structure of the graph.

5.4 Pairs of Vertices

We now consider removing pairs of vertices at a time. Determining which subsets of size two have a positive load effect simplifies the analysis. Additionally, the hope is that if we can analyze these pairs, then we can extend these results to larger subsets. To determine which pairs to consider for a subset we develop some necessary conditions below for the pair to have a positive load effect.

Let G be a graph with key vertex k. Let u and v be two distinct vertices in G not equal to k. Let $\mathcal{E}_k(G, \{u\})$ and $\mathcal{E}_k(G, \{v\})$ be the individual load

effects of u and v, respectively. Then, the following are necessary conditions for $\{u, v\}$ to have a positive load effect on k.

Theorem 5.11. *If $\mathcal{E}_k(G, \{u, v\}) > 0$ and greater than the individual load effects of u and v, then neither u nor v has a single edge-disjoint path to k.*

Proof. Without loss of generality let u have a single edge-disjoint path to k. Consider the graph with v removed. u either has a single edge-disjoint path to k or no path. By Theorem 5.6, removing u has a nonpositive load effect on k. Thus, removing v alone has a greater load effect on k than removing u and v. □

Theorem 5.12. *If $\mathcal{E}_k(G, \{u, v\}) > 0$, $\mathcal{E}_k(G, \{u, v\})$ is greater than the individual load effects of u and v, and u and v are in separate components in $G \setminus k$, then $\mathcal{E}_k(G, \{u\}) > 0$ and $\mathcal{E}_k(G, \{v\}) > 0$.*

For an example, see Figure 5.8.

Proof. Consider the graph G with k removed. Let S_u and S_v be the subsets of vertices in the component of u and v, respectively. Note that $S_u \neq S_v$. Removing u or v cannot increase the number of edge-disjoint paths between vertices in separate components. Furthermore, since these vertices are in separate components when we remove k, these paths must all pass through k and contribute to the load of k.

When we remove a vertex, we decrease the number of these paths between components. Thus, in order for it to be beneficial to remove both u and v, each vertex must increase the number of edge-disjoint paths that pass through k between vertices in their own component. Therefore, each vertex must have a positive load effect on k. □

Theorem 5.13. *If $\mathcal{E}_k(G, \{u, v\}) > 0$ and greater than the individual load effects of u and v, then the list of cycles that contain u cannot be a subset of the cycles that contain v.*

For an example, see Figure 5.9.

Proof. Suppose by way of contradiction that the list of cycles that contain u is a subset of the cycles that contain v. Consider the graph G with v removed. u is no longer involved in any cycles. Thus, u has a single vertex-disjoint path to k. By Theorem 5.7, removing u has a nonpositive load effect on k. Thus, removing v alone has a greater or equal load effect on k than removing u and v. □

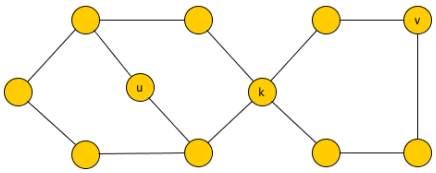


Figure 5.8 An example where u and v are in separate components in $G \setminus k$, $\mathcal{E}_k(G, \{u\}) > 0$, and $\mathcal{E}_k(G, \{v\}) < 0$. Here, k has a load of 46. While u has a load effect of 16, v has a negative load effect of -16. Furthermore, $\mathcal{E}_k(G, \{u, v\}) = 1$. Thus it is not beneficial to remove both u and v.

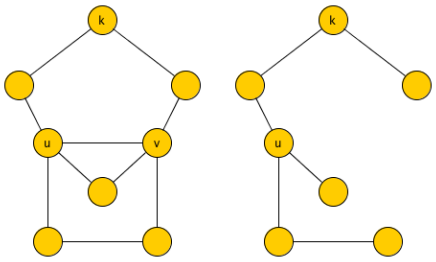


Figure 5.9 An example in which the list of cycles that contain u is a subset of the cycles that contain v. When we remove v, u is no longer in a cycle with k and $\mathcal{E}_k(G, \{u, v\}) \leq \mathcal{E}_k(G, \{v\})$.

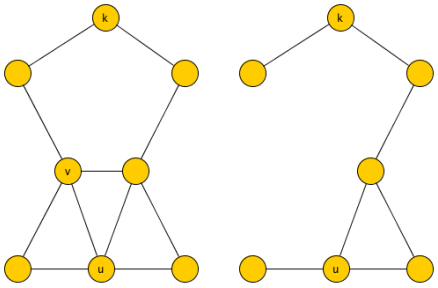


Figure 5.10 An example in which all cycles with v and k contain u. When we remove v, u is no longer in a cycle with k and $\mathcal{E}_k(G, \{u, v\}) \leq \mathcal{E}_k(G, \{v\})$.

Theorem 5.14. *If $\mathcal{E}_k(G, \{u, v\}) > 0$ and greater than the individual load effects of u and v, then all cycles with v and k cannot contain u and vice versa).*

For an example, see Figure 5.10.

Proof. Consider $G \setminus \{v\}$. Suppose in G any cycle containing u and k contained v. Then by removing v these cycles have all been disconnected and u is no longer involved in any cycles with k. By Theorem 5.7, removing u has a nonpositive load effect on k. Thus, removing v alone has a greater or equal load effect on k than removing u and v. □

Theorem 5.15. *If $\mathcal{E}_k(G, \{u, v\}) > 0$ and u and v are not in a chordless cycle together, at least one must have a positive load effect on k.*

Proof. Suppose that $\mathcal{E}_k(G, \{u\}) \leq 0$ and $\mathcal{E}_k(G, \{v\}) \leq 0$. Let $\mathcal{E}_k(G, \{u, v\}) > 0$ and suppose, by way of contradiction, u and v are not in a chordless cycle together.

If we just remove u from the graph, we know that $\mathcal{E}_k(G, \{u\}) \leq 0$. For any pair (s, t) $(s \neq t)$, consider the paths that now must go through k after v is removed from the graph $G \setminus \{u\}$. If this number increases, then $P_{s,t}(\{u, v\}) - P(\{u, v, k\}) > P_{s,t}(\{u\}) - P_{s,t}(\{u, k\})$ where $P_{s,t}(S)$ is the number of edge-disjoint paths between s and t with the subset S removed.

Removing v creates more paths that must go through k. This means that removing v from $G \setminus \{u\}$ decreases the number of edge-disjoint paths between s and t that do not use k. However, since u and v are not in a chordless cycle together, these paths must also be destroyed by removing v in G. Otherwise, this implies a path not using k that goes through v in

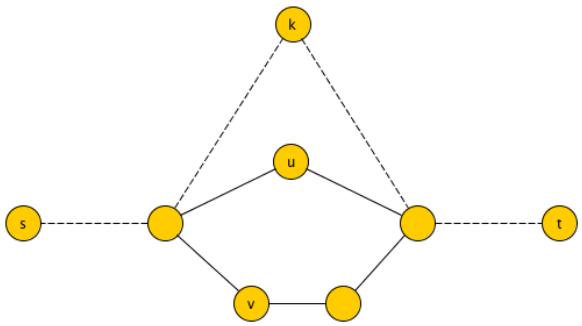


Figure 5.11 An example in which removing u and v increases the number of edge-disjoint paths between s and t that must go through k, but removing u or v alone does not.

G can reroute through u but is destroyed when u and v are both removed, implying u and v are in a chordless cycle together. See Figure 5.11 for an example.

Thus, the load effect of v on k in $G \setminus \{u\}$ is less than or equal to the load effect of v on k in G. Removing v still has a negative load effect and $\mathcal{E}_k(G, \{u, v\}) \leq 0$, which is a contradiction. u and v must be in a chordless cycle together.

□

5.4.1 Implementation

In order to determine when the above conditions hold we can use the following techniques. In Theorem 5.11, we are interested in which vertices have a single edge-disjoint path to k. To determine if a vertex v has a single edge-disjoint path to k, we can calculate the max-flow from v to k in G where each edge has capacity one. If the max-flow is one, then v has a single edge-disjoint path. Otherwise, v has more than one edge-disjoint path to k. This method takes polynomial time in m and n.

In Theorem 5.12, we want to know which vertices are in the same component of $G \setminus \{k\}$. To compute the components, we can run BFS in $H = G \setminus \{k\}$ from a random vertex. All reachable vertices determine one component. We then choose an unvisited vertex and start again. At most we can have n components, so this method is also polynomial in m and n.

Theorems 5.13 and 5.14 refer to the cycles of vertices. To determine if the list of cycles of u is a subset of the cycles of v, we can simply remove

v and search for a cycle with u. This can be done using max-flow where we create a copy of u, u', that is adjacent to all neighbors of u. We then try to push one unit of flow between u and u', but require that we use at least three edges (or the sum of edge flows is greater than two). This is because otherwise we could use construct a cycle (u, x, u'), which is not a cycle in the original graph. If feasible, we have found a cycle with u. Otherwise, there cannot exist a cycle with u in the original graph. Thus, we can determine the solution in polynomial time using linear programming. At most, we need to check $O(n^2)$ pairs so the total time to determine if this condition holds is polynomial in m and n.

In Theorem 5.14, we want to determine if every cycle with k and v contains u and vice versa. To check this, let $H = G \setminus \{v\}$. We then check if u and k are in a cycle. If so, then a cycle with u and k does not contain v. Otherwise, every cycle with v and k contains u. We then repeat and remove u. To check if two vertices u and k are in a cycle, we can use max-flow between u and k where each edge and vertex have capacity one. The vertices are in a cycle if and only if the max-flow is two or greater. At most, we need to check $O(n^2)$ pairs so the total time to determine if this condition holds is polynomial in m and n.

Lastly, in Theorem 5.15, we need to determine if two vertices u and v are in a chordless cycle. To do this, we use a modified network flow linear program in which edges and vertices have capacity one. We then try to push two units of flow between u and v. Let $x_{i,j}$ be the flow across edge (i, j), then we create a new value $y_{i,j}$ for each edge where

$$\sum_{\substack{(i,k)\in E \\ k\neq j}} x_{i,k} + \sum_{\substack{(k,j)\in E \\ k\neq i}} x_{k,j} \geq y_{i,j}.$$

If (i, j) has unit flow across it, $y_{i,j} = 0$ since i cannot have flow to another vertex besides j and j cannot have flow into it besides from i. Otherwise, if i and j participate in flow but $x_{i,j} = 0$, then $y_{i,j} > 0$. Thus, only a chordless cycle has

$$\sum_{(i,j)\in E} y_{i,j} = 0.$$

Let the objective function for our linear program be

$$\sum_{(i,j)\in E} y_{i,j}.$$

By minimizing this objective function, we can determine if u and v are in a chordless cycle in polynomial time, as desired.

Thus, we can determine if the conditions required for Theorems 5.11, 5.12, 5.13, 5.14, and 5.15, in polynomial time for all pairs.

We have now developed some structural characteristics of subsets with a positive load effect on k. These characteristics allow us to eliminate potential vertices or subsets and could be helpful in developing a heuristic. In particular, Theorem 5.15 proves a strong relationship for pairs of vertices with a positive load effect. Future work should be primarily devoted to this area and extending these results to more generalized graphs and subsets. This is discussed in Chapter 6.

Chapter 6

Conclusions and Future Work

In this thesis, we developed tools to increase the amount of communication through a terrorist network's key member, which could potentially increase the visibility of that member.

Using the model presented in Martonosi et al. (2011), this paper demonstrated that LOMAX can be formulated as a mixed-integer linear program. However, given a graph $G = (V, E)$ where $n = |V|$ and $m = |E|$, the program's variables and constraints grow on the order of $O(mn^2)$. This gives us a better understanding of the computational complexity of this problem. As shown in Table 3.1, the model runs out of memory for graphs with more than thirty vertices or around 75 edges.

A Bender's decomposition algorithm was implemented for this mixed-integer linear program. Results again showed that this method is impractical for larger graphs and only generated results for graphs with less than thirty vertices. However, the model and algorithm do show that LOMAX is a linear problem.

In Chapter 5, we developed necessary conditions for vertices to have a positive load effect within a subset. In particular, we examined some simple cyclic structures and pairs of vertices in which the analysis was easier. These properties allowed us to eliminate vertices from consideration and reduce the complexity of the problem. Future work should be devoted to developing new approaches to solving this problem either through other decomposition algorithms or approximation algorithms.

The simplification of the graph and the structural characteristics of vertices and subsets with positive load effect, given in Chapter 5, provide a good start for reducing the complexity of this problem. However, further simplifications need to be made. In particular, analysis of more general

cyclic structures could provide insight into how vertex removals influence the structure of the graphs and which future vertices we might want to remove. Theorem 5.15 could provide a good basis for analysis of larger subsets since it hints that any subset with a positive load effect can sometimes be broken into smaller subsets that each have a positive load effect.

Extension of the theorems in Chapter 5 to more than two vertices would simplify the computations in solving LOMAX, and might be useful for developing any future heuristics.

Bibliography

Bertsimas, D., and J.N. Tsitsiklis. 1997. *Introduction to Linear Optimization*. Belmont, Massachusetts: Athena Scientific, 2nd ed.

Israeli, E., and R.K. Wood. 2002. Shortest-path network indiction. *NETWORKS* 40(2):97–111.

Li, L., D. Alderson, R. Tanaka, J.C. Doyle, and W. Willinger. 2005. Towards a theory of scale-free graphs: Definition, properties, and implication. *Internet Mathematics* 2:4.

Martonosi, S.E., and D.S. Altner. 2009. RUI: Collective research: Algorithms for threat detection: Detecting clandestine members of covert networks. NSF Grant Proposal.

Martonosi, S.E., D.S. Altner, M. Ernst, E. Ferme, K. Langsjoen, D. Lindsay, S. Plott, and A. Ronan. 2011. A new framework for network disruption. Unpublished manuscript.

Motter, A.E., and Y. Lai. 2002. Cascade-based attacks on complex networks. *Physical Review E* 66(6):065,102–065,105.

Royset, J.O., and R.K. Wood. 2007. Solving the bi-objective maximum-flow network-interdiction problem. *INFORMS Journal on Computing* 19:175–184.

Sageman, M. 2004. Understanding terrorist networks. URL http://www.fpri.org/enotes/20041101.middleeast.sageman.understandingterrornetworks.html. Accessed September 20, 2011.