3-2-2007

Heuristically Driven Search Methods for Topology Control in Directional Wireless Hybrid Network

Roger L. Garner

HEURISTICALLY DRIVEN SEARCH METHODS FOR TOPOLOGY CONTROL IN DIRECTIONAL WIRELESS HYBRID NETWORKS

THESIS

Roger Lance Garner, Captain, USAF

AFIT/GCS/ENG/07-03

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

AFIT/GCS/ENG/07-03

HEURISTICALLY DRIVEN SEARCH METHODS FOR TOPOLOGY CONTROL IN DIRECTIONAL WIRELESS HYBRID NETWORKS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Roger Lance Garner

Captain, USAF

March 2007

AFIT/GCS/ENG/07-03

HEURISTICALLY DRIVEN SEARCH METHODS FOR TOPOLOGY CONTROL IN DIRECTIONAL WIRELESS HYBRID NETWORKS

Roger Lance Garner

Captain, USAF

Approved:

Kenneth M Hopkinson	2 Mar 07
Dr. Kenneth M. Hopkinson (Chairman)	Date
Scott R. Graham	2 Mar 07
Maj. Scott R. Graham, Ph.D. (Member)	Date
	2 MAR 07
Dr. Gilbert L. Peterson (Member)	Date

Abstract

Information and Networked Communications play a vital role in the everyday operations of the United States Armed Forces. This research establishes a comparative analysis of the unique network characteristics and requirements introduced by the Topology Control Problem (also known as the Network Design Problem). Previous research has focused on the development of Mixed-Integer Linear Program (MILP) formulations, simple heuristics, and Genetic Algorithm (GA) strategies for solving this problem. Principal concerns with these techniques include runtime and solution quality. To reduce runtime, new strategies have been developed based on the concept of flow networks using the novel combination of three well-known algorithms; knapsack, greedy commodity filtering, and maximum flow. The performance of this approach and variants are compared with previous research using several network metrics including computation time, cost, network diameter, dropped commodities, and average number of hops per commodity. The results conclude that maximum flow algorithms alone are not quite as effective as previous findings, but are at least comparable and show potential for larger networks.

Acknowledgments

I would like to express my sincere appreciation to my faculty advisor, Dr Kenneth Hopkinson, for his guidance, expertise, and support throughout the course of this effort. The creativity and vast knowledge was certainly appreciated. I would also like to thank my committee members, Maj. Scott Graham and Dr. Gilbert Peterson for their fervent advice and assistance throughout this process. Last, but certainly not least, I wish to thank my lovely wife for her unwavering support and compassionate understanding as I set and surpass this prestigious milestone in my life.

Roger Lance Garner

Table of Contents

Page

Abstract iv
Acknowledgments v
Table of Contents vi
List of Figures ix
List of Tables xii
I. Introduction 1
Background 1
Problem Statement 3
Research Approach 4
Research Objectives/Questions/Hypotheses 5
Preview 6
II. Literature Review 8
Chapter Overview 8
Background 8
Flow Networks and Network Flows 11
Maximum Flows 14
Multi-Commodity Flows 18
Relevant Research 20
Summary 26
III. Methodology 27
Chapter Overview 27
Problem Example 27

Choosing the Combination 31
Knapsack 32
Greedy Technique 34
Solving the Combination 35
Edmonds-Karp 35
Pre-flow Push 39
Potential Edges 44
Summary 45
IV. Analysis and Results 47
Chapter Overview 47
Design of Experiments 47
Test Computer Specifications 49
Order Complexity 49
Limitations 50
Results of Experiments 52
Metric 1: Run time 52
Metric 2: Number of hops 57
Metric 3: Dropped commodities 61
Metric 4: Total Cost 65
Metric 5: Network diameter 69
Summary 73
V. Conclusions and Recommendations 75
Chapter Overview 75

Brief Review....75
Conclusions of Research....76
Recommendations for Future Research....77
Summary....80
Appendix A....81
Bibliography....84
Vita....87

List of Figures

Page

Figure 1. Implication of Information Superiority as reflected in JV 2020[4]. 2

Figure 2. Routing and Topology Illustrated[5] 6

Figure 3. Autonomous Reconfiguration Process[17]. 11

Figure 4. An example of a flow network. 13

Figure 5. Greedy method for maximum flow finds max flow = 10. 15

Figure 6. Non-greedy method for maximum flow finds the optimal max flow = 15. 15

Figure 7. Multi-source, Multi-sink flow network. 19

Figure 8. Super source, super sink transforms the network into a single-source, single sink flow network. 19

Figure 9. Comparison of Erwin's MILP method[13] and Kleeman's MOEA method[9] for a 10-node network. 25

Figure 10. 4-node network with 2 unique interfaces at each node. 28

Figure 11. Dynamic Knapsack Pseudocode[32]. 33

Figure 12. General Edmonds-Karp Maximum Flow Algorithm[24]. 36

Figure 13. Updated Edmonds-Karp Algorithm. 39

Figure 14. General Pre-flow Push Max Flow Algorithm[23]. 40

Figure 15. Updated Pre-flow Push Algorithm. 44

Figure 16. Potential edge illustration. 45

Figure 17. Methods for solving the problem. 46

Figure 18. The difference between n and n^2 commodities. 51

Figure 19. Run time (s) for each method on a 10-node network. 53

Figure 20. Run time (s) for each method on a 15-node network. 54

Figure 21. Run time (s) for each method on a 20-node network (logarithmic scale). 54

Figure 22. Run time (s) for each method on a 25-node network. 54

Figure 23. Run time (s) for each method on a 30-node network. 55

Figure 24. Overall run time performance for all tested networks. 56

Figure 26. Average number of hops for each method on a 15-node network. 58

Figure 28. Average number of hops for each method on a 25-node network. 59

Figure 29. Average number of hops for each method on a 30-node network. 59

Figure 30. Overall average number of hops for all tested networks. 60

Figure 31. Number of dropped commodities for each method on a 10-node network (total possible = 90). 61

Figure 32. Number of dropped commodities for each method on a 15-node network (total possible = 210). 62

Figure 33. Number of dropped commodities for each method on a 20-node network (total possible = 380). 62

Figure 34. Number of dropped commodities for each method on a 25-node network (total possible = 600). 63

Figure 35. Number of dropped commodities for each method on a 30-node network (total possible = 870). 63

Figure 36. Overall average number of dropped commodities for all tested networks. 65

Figure 37. Total cost broken out by link and fixed cost for each method on a 10-node network. 66
Figure 38. Total cost broken out by link and fixed cost for each method on a 15-node network. 66
Figure 39. Total cost broken out by link and fixed cost for each method on a 20-node network. 67
Figure 40. Total cost broken out by link and fixed cost for each method on a 25-node network. 67
Figure 41. Total cost broken out by link and fixed cost for each method on a 30-node network. 68
Figure 42. Overall total cost for all tested networks. 69
Figure 43. Network diameter for each method on a 10-node network. 70
Figure 44. Network diameter for each method on a 15-node network. 70
Figure 45. Network diameter for each method on a 20-node network. 71
Figure 46. Network diameter for each method on a 25-node network. 71
Figure 47. Network diameter for each method on a 30-node network. 72
Figure 48. Overall network diameter for all tested networks. 73
Figure 49. Combination of multiple methods used together. 79

List of Tables

Page

Table 1. LP Formulation - Variables of Interest[13] 21

Table 2. List of commodities for the example network. 29

Table 3. Edge list for the example network. 29

Table 4. Metrics used for comparison with Erwin's results. 48

Table 5. Average performance statistics for each method on a 10-node network. 81

Table 6. Average performance statistics for each method on a 15-node network. 81

Table 7. Average performance statistics for each method on a 20-node network. 82

Table 8. Average performance statistics for each method on a 25-node network. 82

Table 9. Average performance statistics for each method on a 30-node network. 83

Table 10. Average performance statistics for each method on a 35-node network. 83

HEURISTICALLY DRIVEN SEARCH METHODS FOR TOPOLOGY CONTROL IN DIRECTIONAL WIRELESS HYBRID NETWORKS

I. Introduction

Background

On the 64th anniversary of the attack on Pearl Harbor (7 December 2005), the Secretary of the Air Force, Michael W. Wynne and Air Force Chief of Staff, General T. Michael Moseley announced an important change in the Air Force mission statement[1, 2]. The new statement read "The mission of the United States Air Force is to deliver sovereign options for the defense of the United States of America and its global interests--to fly and fight in Air, Space, and Cyberspace." One of the major changes was the incorporation of a new battlefield domain: Cyberspace. Steps to "operationalize" a new Cyber Command have already begun[3]. This change is directly in line with the greater goals outlined in the Department of Defense's (DoD) Joint Vision (JV) 2020[4], which highlights the importance of not only achieving, but maintaining information superiority across the full spectrum of joint military operations as depicted in Figure 1.

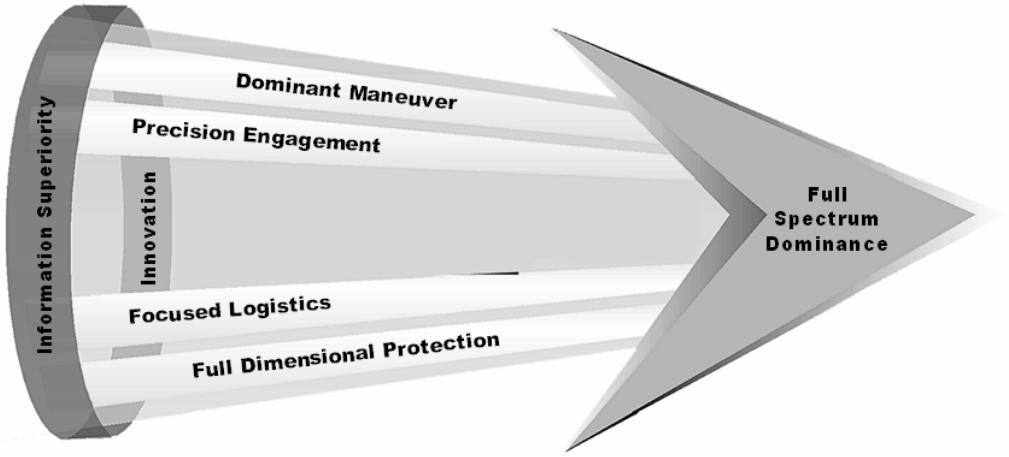

Figure 1. Implication of Information Superiority as reflected in JV 2020[4].

The leaders of our Armed Forces have voiced the significance of taking proactive steps necessary in achieving a dominant role in Cyberspace. Thus, the research and educational community must spearhead efforts to develop the means to realize this goal. A major catalyst in these regards is the proliferation of wireless communications technology and wireless networks.

While general research for commercial wireless applications is quite mature, focus on the new domain of cyberspace as it pertains to the military is in its infancy. This is due in part to dissimilar requirements from the non-military community.

The research described in this document takes a Net-Centric Warfare approach to solving a small part of a much larger research endeavor: Hybrid Communications Network Control and Management. The fundamental goal of this group is the creation of a basis for "a realistic infrastructure to support Net-Centric Warfare in directional wireless networks with a hybrid (i.e., heterogeneous) mixture of directional free space

optimal and radio frequency (RF) devices[5].” Given a suitable control structure, a combination of directional and omnidirectional interfaces in hybrid networks promise greater bandwidth, more flexibility, and lower latency than today’s homogeneous omnidirectional networks. These ideas are explained in more detail throughout the document.

Problem Statement

A key area of research in Hybrid Communications is the Topology Control Problem (also known as the Network Design Problem). This domain is briefly defined as determining a feasible network topology. A network is defined as a set of nodes and a set of links. Some examples of network nodes are servers and routers. Fiber optic wires, the radio frequency (RF) medium, and Free Space Optical (FSO) links are examples of links (also referred to as edges or arcs). Given a network and a set of traffic requirements (i.e., data packets traversing the network), the objective is to determine the desired physical connectivity of the network such that an optimum subset of traffic requirements can be met according to a pre-defined set of criteria (e.g., minimal cost and maximum throughput). In this definition, feasibility can have multiple meanings. For example, given the potentially extreme consequences of a compromised military network, administrators may place more importance on network characteristics such as low probability of detection/low probability of interception (LPD/LPI). On the other hand, a network relying on time-sensitive data communications would likely be more interested in a network topology with the shortest average delay. Each of these characteristics, discussed in subsequent chapters, contribute to the problem difficulty.

Nonetheless, the primary goal of obtaining a feasible topology is constrained by the number nodes in the network, the available links and link properties, and the demand of traffic requirements (also referred to as commodities). Solution elegance lies in properly balancing these constraints with the requirements of the user.

Research Approach

Calculating efficient and effective topologies for wireless networks is a difficult problem. A number of research articles have appeared as of late looking at how to create topologies in wireless networks consisting of omnidirectional RF transmitters[6], but topology control algorithms for directional links, such as directional RF or laser links, have received much less attention. While current wireless technology is largely omnidirectional, particularly in consumer products, omnidirectional nodes have poor range and known scalability problems[7]. To circumvent these issues, future large-scale military networks are likely to use a mixture of directional and omnidirectional links.

Topology control using directional links is an NP-hard problem[8]. Thus, research looking at directional topology control centers on the development of heuristics, which provide suboptimal, but timely solutions, and on integer linear programming methods, which provide an exact solution if the problem size is kept to a relatively small size. Other methods discussed are based on searching the solution space for feasible answers using techniques such as evolutionary computation[9] and other informed search strategies[10, 11].

Within this domain, the quality of the topology T is able to be evaluated according to several criteria including connectivity, energy-efficiency, throughput, and robustness to

mobility, etc[12]. For the purposes of this research, two primary criterions are used. The first is the solution's run time. 21st century military networks must be able to adapt their communication topologies according to unexpected changes in network demand, unit positions or device locations, and link interference patterns. Therefore, topology control algorithms must provide near real time solutions

The secondary criterion is the quality of the solutions generated by the algorithms. Solution quality helps determine effectiveness of the algorithm itself. Quality in this research is measured primarily via cost. Applications require a sound infrastructure to maintain operational networks supporting military operations. If solutions are generated within acceptable time constraints, yet suffer from poor quality—such as a partitioning of the network—then feasibility has not been achieved.

Research Objectives/Questions/Hypotheses

The objective of this research is to create and analyze the efficiency and effectiveness of search methods that have a relatively low, i.e. polynomial, computational complexity while delivering an acceptable quality of service (QoS). To do this, our model is based on flow networks and maximum flow algorithms. This genre of networks embodies certain positive characteristics that help achieve the objectives outlined above. These characteristics empower the communications networks employed by the military to demonstrate a high degree of flexibility that is demanded by the information-driven community.

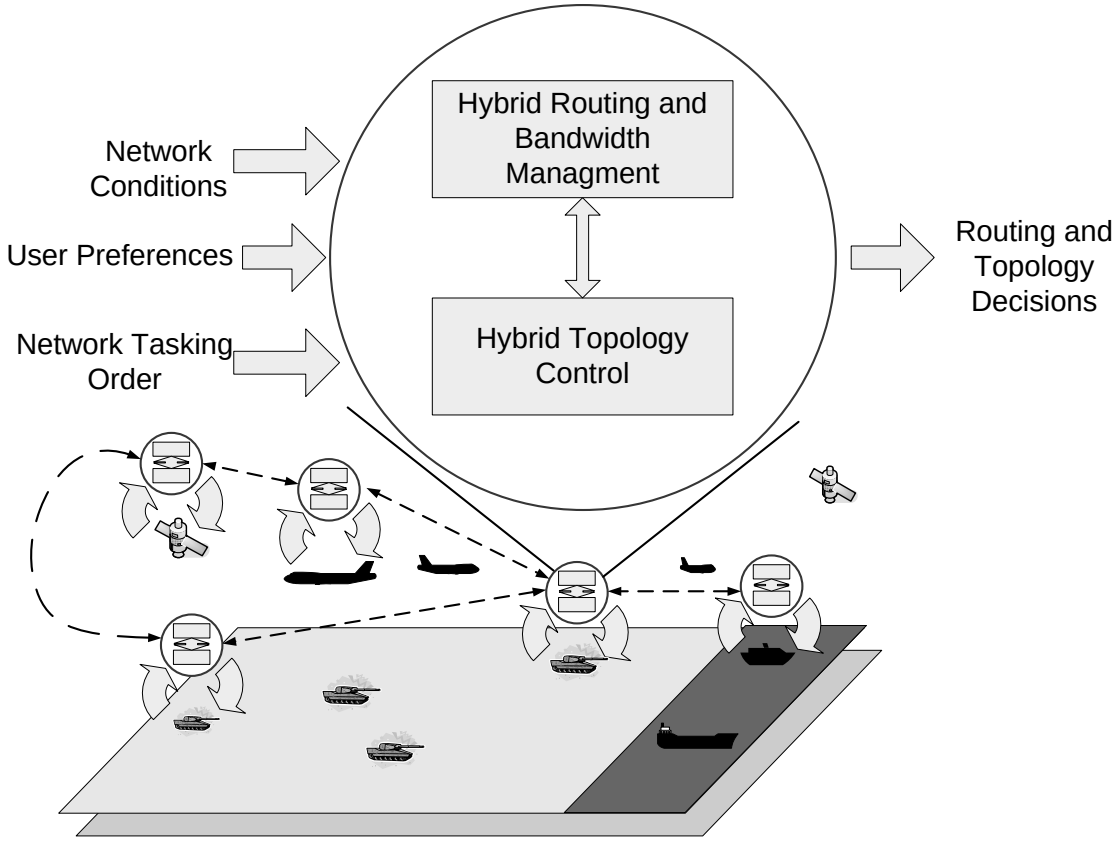

Figure 2. Routing and Topology Illustrated[5].

Preview

This chapter provides a general introduction to the topology control problem, briefly outlines the importance of the research and its objectives, and provides a short preview of the remaining topics covered in this document. Chapter Two (II) introduces the reader to the general area of flow networks and presents an overview of other research efforts that are of significant relevance to the problem statement. Chapter Three (III) details the methodology and approach used during this endeavor including a complete problem definition and explanation of the techniques employed. Chapter Four (IV) explicates the experimental design results and discusses the challenges with creating

and executing them. Finally, Chapter Five (V) captures the essence of this research effort by summarizing the key points, discussing the observations and conclusions made, and outlining ideas for subsequent research.

II. Literature Review

Chapter Overview

This chapter introduces the relevant research and literature focused on topology control and wireless network communications. First, a background on topology control is provided. Second, a brief review of network flows is presented as it is the basis for the methodology discussed in Chapter III. Lastly, previous research is examined. Of particular interest is the research of [13]. This research was the original motivation behind the studies described in this document and helps illustrate the problem in more detail.

Background

In laymen's terms, the topology control problem translates to making a decision about how to connect the network to maximize performance. As the technology matures, it will soon become common practice for networked devices to have multiple connection interfaces and/or the ability to connect to one of several potential destination nodes. Networks are normally not scheduled at 100% of their capacity in order to promote stability and fault tolerance. In some instances it may even be practical to save links if they are not required. This is especially important when dealing with wireless sensor networks as power consumption is a primary concern[14]. The choice of topology, given the vast possibilities, can greatly affect the performance of the network.

According to [15], "topology control is one of the most important techniques used in wireless networks." Performance in this sense can be thought of as traffic

throughput—maximizing the amount of traffic that can be traversing the network at any given time within the bounds imposed by the hardware infrastructure.

Routing traffic in a network is related to topology control, but is a simpler problem. Even so, effectively routing traffic can be a difficult problem when attempting to manage multiple datastreams, each with its own QoS considerations. With simple networks (i.e., single source and/or single destination) this problem can be solved relatively easily using a variety of maximum flow algorithms, described later. However, the problem becomes significantly more complicated when considering multiple traffic sources and destinations, especially when a node exhibits characteristics of both.

As an example, consider a small group of unmanned aerial vehicles (UAVs) hovering over the battlefield. One principal topic in research dealing with these aircraft is swarming (clustering of UAVs in such a manner that mimics the behavior occurring in nature of, for example, colonies of bees or ants, or schools of fish[16]). To effectively imitate a swarm, these aircraft must communicate amongst one another. If, for example, a particular UAV is unable to communicate directly with another UAV, the flow of information must be routed via the swarm, much like a typical communications network. With the aircraft constantly moving, there is a high probability that links will fail. However, as the aircraft change position, new links will become available.

With infrastructure-based networks (or static networks) topology control is much less complex over the life of the network than is the case in mobile situations. Once a solution is found and implemented in fixed networks, no other work is required. However, with mobile networks, many solutions may be needed over time. Therefore,

the network must continually adapt its topology to accommodate for state and topographical changes in order to maintain efficiency.

Topology control can be thought of as an autonomous reconfiguration process[17]. Over the life of the network, this process is continually repeated. There are five states in this process:

1) Link state examination,

2) Collection of link state information,

3) Solution computation,

4) Solution distribution, and

5) Reconfiguration

During the first step, the network state is examined to determine if a change has occurred. Here, each node must track local topological data and collect traffic statistics such as adjacent node location changes, additions and deletions of nodes, increased traffic load, etc. This information is then forwarded to a controller, which analyzes the data of the entire network. If the data collected reveals that a topology change is required, control moves to step three. At this stage, the controller computes a new solution. Ideally, it would not only find an optimal solution in terms traffic requirements, but it would also satisfy the other constraints of the network, such as resolving heavily congested nodes, minimizing the overall usage of available resources (i.e., links, capacity, etc), and it would consider user preferences. Once a solution has been computed, two things must happen. The solution must be redistributed among the community of nodes, and each node must implement the new topology. This may

include the selection of new links or the redirection of communication devices, such as laser beams, using other mediums for certain traffic while the re-orientation of directional links is taking place.

As you can imagine, with highly mobile networks, this process must be very efficient—on the order of a few seconds if not faster depending on the size and dynamics of the network. Figure 3 illustrates the process of autonomously reconfiguring a network. The primary focus of this research is within phase three: computing the solution.

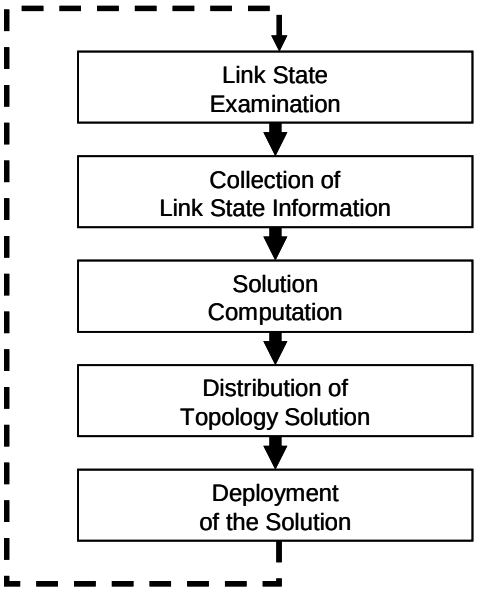

Figure 3. Autonomous Reconfiguration Process[17].

Flow Networks and Network Flows

Flow networks are a generalization of communications networks. They are called such because the traffic that the network accommodates is a network flow. A network flow is "an abstract entity that is generated at source nodes, transmitted across edges, and

absorbed by sink nodes[18].” This concept is synonymous with data packets on a network, also called commodities. Throughout this document the terms network flow, traffic, and commodity are used interchangeably. The following formulation is based on the flow networks definition.

A flow network is defined as a graph $G = (V, E)$ where V is the set of vertices (or nodes) and E is the set of edges (or links, arcs). Each edge contained in the set E is both directional and capacitated. Furthermore, set V contains two particular nodes, s and t, which represent the source and sink, respectively. The source node is the origin of all traffic within the flow network. It has a “flow reservoir” with unlimited capacity, which simulates a continuous flow. The sink represents the opposite. All flow exiting the source must eventually enter the sink, as it is the destination of all flow within the network. All other nodes are simply intermediate, but are equally as important[18].

To really understand the uniqueness of flow networks, we turn to the three necessary properties that must be satisfied at all times. They are:

1) Skew Symmetry:

$$\forall u, v \in \mathrm{V}: \quad f(u,v) = -f(v,u) \tag{1}$$

2) Capacity Constraint:

$$\forall e \in \mathrm{E}: \quad 0 \leq f(e) \leq c(e) \tag{2}$$

3) Flow Conservation:

$$\forall v \in \mathrm{V} - \{s, t\}: \quad \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e) \tag{3}$$

Skew symmetry implies that the flow on an edge must equal the negative flow of the edge in the opposite direction. The capacity constraint simply requires that the flow on an edge must not exceed the capacity of the edge. Finally, flow conservation states that the flow coming into a node must equal the flow exiting the node. Figure 4 illustrates a simple flow network without flow on its edges.

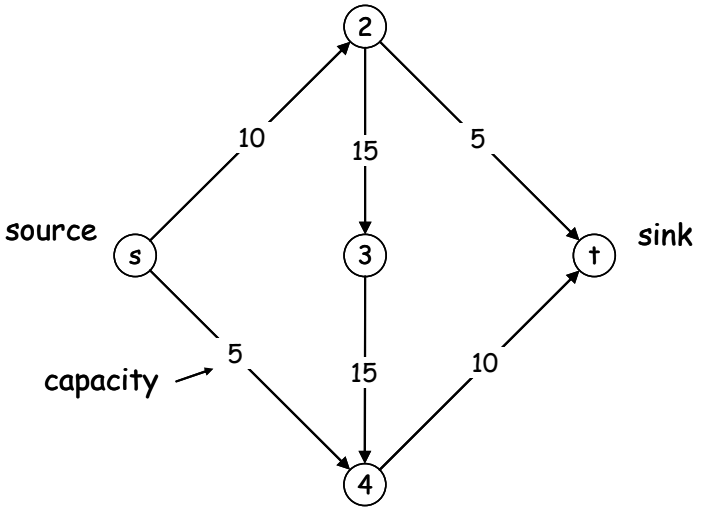

Figure 4. An example of a flow network.

Network flows are applied to a variety of problems such as the shortest path problem, assignment problem, transportation problem, and minimum cost flow problem to name a few. However, this research assumes a capacitated network in which attempts are made to maximize commodities. Therefore, it is appropriate to model the topology control problem after maximum flow problems. The following provides a brief description of maximum flow problems and how to solve them.

Maximum Flows

Maximum flows date back to the mid 1950s when Russian scientist A. N. Tolstoi investigated the soviet rail network in an effort to optimize the amount of cargo that could be shipped from within the Soviet Union to destinations located in westerly satellite counties[19]. By modeling the railroad network as a flow network, the Russians were able to calculate optimal routes and identify single points of failure.

The basic premise of maximum flows is to answer the question "How much flow can be transferred from the source node to the sink node while satisfying the constraints imposed by the flow network?" Although much literature exists explaining methods for solving maximum flows, generally speaking they fall into two categories of algorithms: augmenting path and pre-flow push[18, 20].

An augmenting path is defined as a set of ordered edges from source to sink where the capacities on each of the edges in the path are positive. Algorithms that utilize this method include some greedy algorithms such as the Ford-Fulkerson method[18, 20, 21] and the Edmonds-Karp algorithm[18, 20-22].

Greedy algorithms use an elitist approach to order the augmenting paths found. At each step in the algorithm, the edge with the highest capacity is taken next. As often seen with greedy algorithms for any problem, greedy maximum flow algorithms can get trapped in a local minimum. That is to say that upon algorithm termination, the solution cannot be guaranteed to be optimal, and in fact, often times it is not. Figure 5 and Figure 6 illustrate this potential drawback.

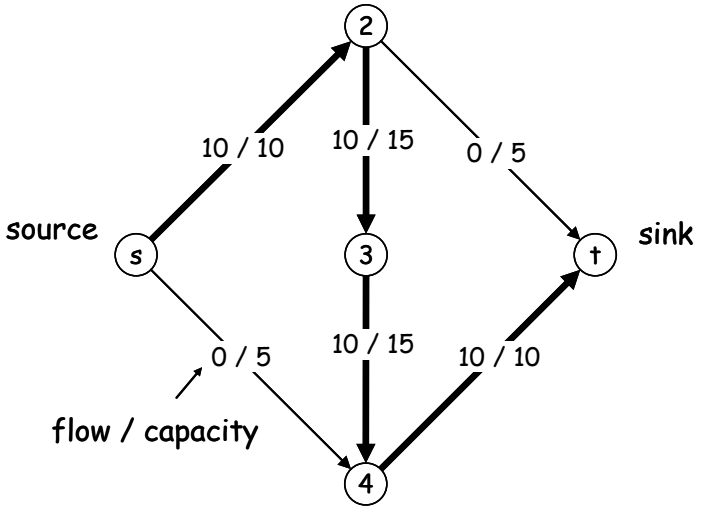

Figure 5. Greedy method for maximum flow finds max flow = 10.

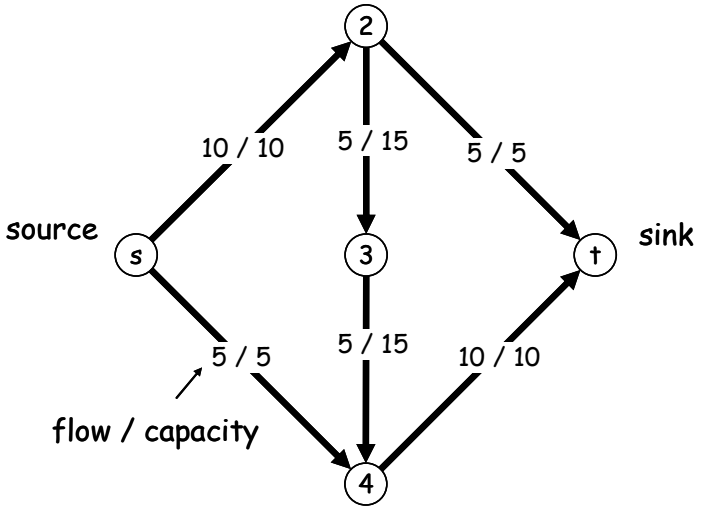

Figure 6. Non-greedy method for maximum flow finds the optimal max flow = 15.

The Ford-Fulkerson method[18, 20] is another member of the augmenting path family of maximum flow algorithms. This algorithm uses what is known as a residual network to keep track of flow that has already been pushed on the edges. For example, as flow x is added to edge $a \rightarrow b$ in the original network, flow $-x$ is added to edge $b \rightarrow a$ in

the residual network. This creative technique allows flow to be redirected without disrupting previous flows that have already been solved. In the general case, augmenting paths are arbitrarily chosen. That is, the implementation of the data structures determine how subsequent paths are chosen (i.e., depth first, etc.). Not only does this requires tedious bookkeeping, but if proper consideration for choosing augmenting paths is disregarded, the performance of the algorithm overall suffers (reference pp 352 - 353 of [18] for a specific example). If we let f^* be the maximum flow value calculated, then [20] states that the Ford-Fulkerson method exhibits an order complexity of $O(E \times f^*)$. In fact, it is for this very costly reason that we examine the next maximum flow algorithm.

The Edmonds-Karp algorithm is a based on the Ford-Fulkerson method described above. The primary difference is that during the process of choosing an augmenting path, rather than arbitrarily selecting the next hop node, it implements a breadth-first search. The resultant augmenting path is then provably the shortest path from source to the sink among the paths that have not been chosen (i.e., for every path p in P, $p_i < p_{i+1}$). This adjustment leads to a more desirable order complexity of $O(V \times E^2)$[18, 20, 23]. A further explanation of this algorithm is provided in the next chapter.

The maximum flow algorithms previously discussed are rather simplistic in their approach to solving the problem. The second class of these algorithms is the Pre-flow Push[23]. These algorithms take a local approach to deriving the maximum flow using the notion of nodes within a hierarchy. In addition, they do not utilize augmenting paths like the previous algorithms, which unfortunately make them difficult to comprehend. There are a number of variations such as the push-relabel algorithm and relabel to front

algorithm[18, 20, 24]. In general, these algorithms center around three basics concepts: the pre-flow, the push operation, and the pull operation.

Kleinberg and Tardos[18] define a pre-flow as "a function *f* that maps each edge *e* to a nonnegative real number, $f: E \rightarrow \boldsymbol{R}^{+}$" in which the flow conservation constraint is modified such that the flow into a particular node is greater than or equal to ("≥") the flow out of that node (rather than equal to, i.e., "="). Upon completion of the algorithm, flow conservation condition is preserved, thus transforming the pre-flow into a flow.

The push and pull operations focus on the concept of getting flow to the sink from a given node and receiving flow from the source. In the general case, a node is selected and flow is recursively pushed to the sink along available "outgoing" edges as well as recursively pulled from the node's neighbors via available "incoming" edges. This concept is significantly different than finding an augmenting path from the source to the sink.

Using these concepts, [23] concludes that an order complexity of $O(V^3)$ can be obtained, which is provably better than the Edmonds-Karp algorithm. This is especially I important when the network is very dense as run time experienced by the Edmonds-Karp algorithm is a factor of the number of edges.

Note that neither of these algorithms incorporates potential edges, a unique approach in this research. However, with a few modifications this can be easily resolved. In addition, these two algorithms are unique in their own sense. When compared to Pre-flow Push, the Edmonds-Karp algorithm is much easier to comprehend and implement. Thus, while in theory the Pre-flow Push algorithm has potential for better solutions both

it and the Edmonds-Karp algorithm are ideal candidates for application to the topology control problem.

Multi-Commodity Flows

One of the major drawbacks of maximum flow algorithms is that they were designed with single-source, single-sink networks in mind. Thus, the source node has no incoming edges and the sink node has no outgoing edges. This is typical of simple flow networks, however, as you introduce multiple traffic patterns, any node has the potential of being a source node, sink node, intermediate node, or any combination thereof with respect to different flows traveling across the network. To circumvent this, consider the following example.

One solution for transforming single-source, single-sink networks into multi-commodity flow networks is to implement a super source and a super sink, described in [20]. The super source s is a special node that is directly connected to each source s_i via a directed edge e with infinite capacity—or at least the capacity of the source in which it is connected—from s to s_i (i.e., for each source s_i in V, edge $s \rightarrow s_i$ exists in E). Similarly, the super sink t is a special node that is directly connected to each sink t_i via a directed edge e with infinite capacity (or at least the capacity of the sink in which it is connected) from t to t_i (i.e., for each sink t_i in V, edge $t_i \rightarrow t$ exists in E). Figure 7 and Figure 8 illustrate this concept.

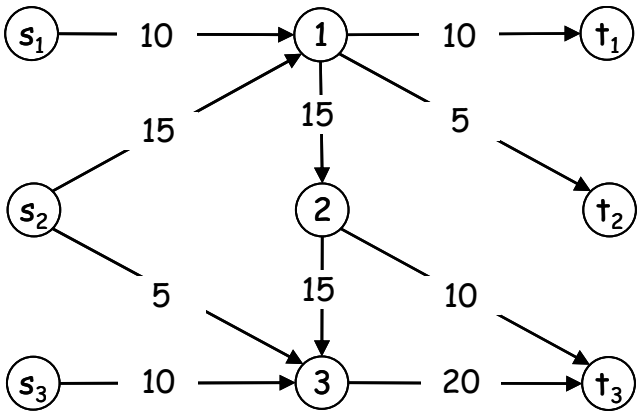

Figure 7. Multi-source, Multi-sink flow network.

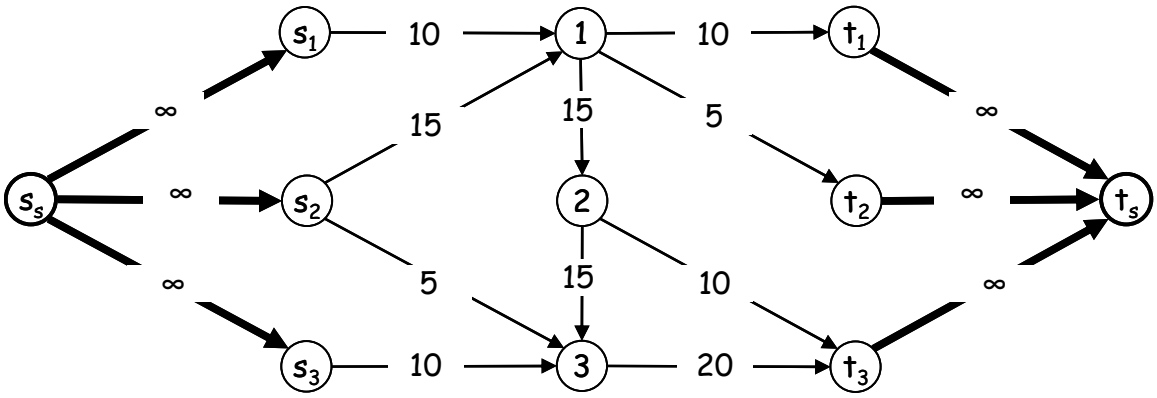

Figure 8. Super source, super sink transforms the network into a single-source, single sink flow network.

This suffices for maximum flow calculation, but it doesn't permit the determination of which commodities are satisfied because it transforms the problem from multi-commodity to single-commodity. This is not desirable as real world commodities may very well require preemption or prioritization. Also, in many cases, a demand must be satisfied completely to be useful. For example, receiving only half of a zip file is not helpful as there will be no way to read it. For this reason, we turn to the knapsack

formulation—a well known routine for determining the most valuable set of goods (i.e., commodities) given the constraint of a cost function. This concept is further explained in the next chapter.

Relevant Research

This section provides a brief introduction to the current research within the Network Design Problem (NDP) domain. While there are distinctive variants of the NDP, this research is mainly concerned with the multi-commodity capacitated NDP (MCNDP). This variation is well-suited for real world simulations because of the characteristics it models. As its name implies, the MCNDP constrains edge capacities and handles multiple commodities. One particularly interesting formulation is that of [13], which is described below.

Erwin uses a Mixed-Integer Linear Programming (MILP) approach to solving the NDP. MILP techniques provide an optimal solution[24], but one which is very costly with respect to time. Erwin's research validated this concept with networks of size $n \geq$ 15 nodes. In fact, for a network of 15 nodes, the average time required to compute the solution via MILP methods was ~13 minutes[13]. It is reasonable to expect that future mobile real world networks will larger and will require a much faster response time.

The primary objective of Erwin's research was to minimize the link and flow cost of a network topology subject to a variety of constraints, such as bandwidth, interfaces, degree of arcs (i.e., links or edges). His model provides an optimal solution of the topology control problem, however as we briefly stated above, there is one major

drawback—running time, albeit with excessive runtime. As the number of nodes, edges, interfaces, and commodities increased, so does the complexity.

The mathematical formulation is now presented including the variables of interest (Table 1), objective function, and problem constraints.

Table 1. LP Formulation - Variables of Interest[13].

Variable	Definition (representation)
N	set of nodes
K	number of commodities
F	number of interface types
(i,j,f)	arc connecting node *i* to node *j* by interface type *f*.
A	node-incidence matrix where $a_{ijf} = 1$ if node *i* is incident to node *j* via interface type *f*, and 0 otherwise.
x_{ijf}^{k}	fraction of the required flow of commodity *k* to be routed from the source (s^k) to the destination (d^k) that flows on arc *(i,j,f)*
y_{ijf}	binary variable indicating whether arc *(i,j,f)* is selected as part of the network topology
v_{ijf}^{k}	per unit cost for commodity *k* on arc *(i,j,f)* multiplied by the flow requirement for that commodity
c_{ijf}	fixed cost of including arc *(i,j,f)* in the network
u_{if}	number of interfaces of type *f* at node *i*
b^k	required bandwidth for commodity *k*
cap_{ijf}	the capacity of arc *(i,j,f)*

The objective function identifies what is trying to be optimized. In Erwin's research, the specific objective was to minimize the total cost of the network. Total cost is the sum of two cost variables. The first, link cost, is the fixed cost to use a given link in the network. The second, flow cost, equates to the cost to route a given commodity across a particular link. Recalling the variables listed in Table 1, the objective function is formally stated as:

Minimize:

$$\sum_{\{k,(i,j,f):a_{ijf}=1\}} v_{ijf}^{k} x_{ijf}^{k} + \sum_{\{(i,j,f):a_{ijf}=1\}} c_{ijf} y_{ijf} \tag{4}$$

The first summation totals the cost of per unit of flow for each commodity on each edge. The second summation totals the cost of each edge included in the final topology. Calculating the objective function seems rather simple at this point. What has yet to be considered, however, are the constraints.

Constraints provide limits on how a problem is solved. They are necessary to ensure certain criteria remain within acceptable parameters. Calculating the objective while adhering to a plethora of constraints is what makes optimization problems difficult. Erwin's objective function was subject to the following constraints:

$$\sum_{j,f:a_{ijf}=1} x_{ijf}^{k} - \sum_{j,f:a_{jif}=1} x_{jif}^{k} = \begin{cases} 1 \text{ if } i = s^{k} \\ -1 \text{ if } i = d^{k} \\ 0 \text{ otherwise} \end{cases} \quad \forall i \in N, k = 1,...,K \tag{5}$$

Equation 5 constrains the amount of commodity *k* on link *(i,j,f)* such that it can not exceed 100 percent.

$$\sum_{k} r^{k} x_{ijf}^{k} \le cap_{ijf} \qquad \forall (i,j,f) \in A, \ni \ a_{ijf} = 1 \tag{6}$$

Equation 6 ensures the sum of the flows for all commodities across link *(i,j,f)* is no greater than the capacity if the link itself.

$$\sum_{j \in N} y_{ijf} \le u_{if} \qquad \forall i \in N, f = 1,...,F \tag{7}$$

Equation 7 implies that the number of links from node *i* to node *j* is no larger than the number of interfaces at that node.

$$x_{ijf}^{k} \le y_{ijf} \qquad \forall (i,j,f) \in A \ni a_{ijf} = 1, k = 1,...,K \tag{8}$$

Equation 8 guarantees not only that the amount of commodity *k* on link *(i,j,f)* is at most one, but also ensures that if the link is not included in the topology, then no commodity can be routed across it.

$$y_{jif} = y_{ijf} \qquad \forall (i,j,f) \in A \ni a_{ijf} = 1 \tag{9}$$

Equation 9 assures that if there is a link *(i,j,f)* then there must be a link *(j,i,f)* in the other direction as well.

$$x_{ijf}^{k} \ge 0 \qquad \forall (i,j,f) \in A \ni a_{ijf} = 1, k = 1,...,K \tag{10}$$

Equation 10 implies that the percentage of a commodity across a given link must be positive.

$$y_{ijf} \text{ is binary} \qquad \forall (i,j,f) \in A \ni a_{ijf} = 1 \tag{11}$$

Lastly, equation 11 constrains the decision to use link *(i,j,f)* in the network to be 0 or 1.

Once the model has been formulated, it can be integrated into a linear solver. Erwin used Xpress-Optimizer, a component of the Xpress-MP suite and a well-known software optimizer for integer-linear programming problems, as his solver[25]. One benefit gained from this software is the built-in implementations of multiple popular algorithms. Specifically, Erwin was able to run simulations using three different methods: Primal Simplex, Dual Simplex, and Newton Barrier[13, 26]. Erwin discovered that while these methods provide high quality solutions, they scale poorly relative to

network size. To compensate, he proposed three additional methods to take advantage of the MILP optimality while circumventing its complexity weaknesses.

These additional methods each have a common theme: the degree-constrained minimum spanning tree (dcMST). The dcMST provides a preliminary network that ensures connectedness. From there, edges are added to form the resultant topology via one of three methods. The first two approaches add edges to nodes in non-decreasing order or non-increasing order. Erwin refers to these methods as heuristic 1 and heuristic 2[13]. The third "edge-adding strategy" utilizes the MILP formulation described above. This method is referred to as the "combo" method because it uses a combination of the MILP formulation and the dcMST. While these methods do not guarantee optimality, they exhibit significant reductions in runtime. These results are examined further in Chapter IV.

Kleeman, et al.[9] developed a multi-objective evolutionary algorithm (MOEA) to solve the same problem. Motivation for implementing a stochastic algorithm comes from the exponential growth as the problem size increases. They use the same formulation provided by Erwin described above. Genetic algorithms in general are considered to be somewhat slow (due to the number of generations that must occur), thus, it is not surprising that their simulations took roughly three minutes per trial versus ~19 seconds for Erwin's MILP trials. However, it is interesting to note that their technique saw improved performance in total cost for each of 10 trial runs considering that the MILP method should provide the optimal solution, suggesting either a relaxation of constraints or improper formulation. Figure 9 highlights their results.

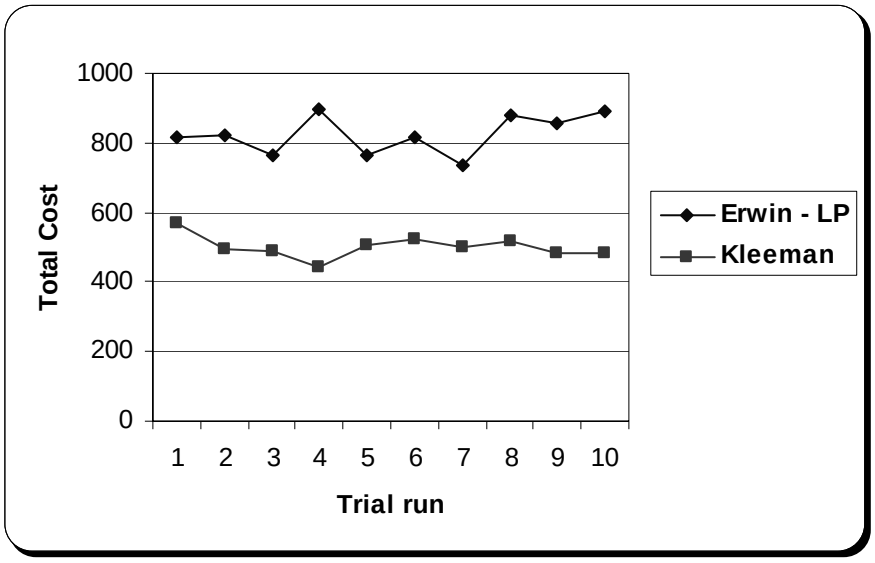

Figure 9. Comparison of Erwin's MILP method[13] and Kleeman's MOEA method[9] for a 10-node network.

Desai and Milner[27] propose a number of "scalable congestion minimization heuristics" which they apply to ring topologies. Their techniques are summarized here. The "single-hop" heuristic tries to maximize the flows of commodities where the source and destination are one-hop neighbors. The "multihop" heuristic attempts to establish the least congested, multihop path for commodities in which the source and destination are not directly connected. These two heuristics demonstrate an order complexity of $O(V^3)$. The "rollout" method works by sorting all K commodities in non-decreasing order. It then creates K topologies using either of the heuristics described above such that the k^{th} topology is more conducive to solving commodity k before any other commodities (i.e., topology $k = 2$ is created with the commodity order { 2, 0, 1, 3, 4, …, K - 1 }). The fourth technique, "Branch Exchange," creates approximately V^2 topologies by

enumerating possible combinations of exchanging two links currently in use with two links that are not in use. The latter two methods are both bounded by $O(V^5)$. Further research is expected for networks with more than two degrees of connectivity.

Finally, Davis, et al.[6] take a clustering approach to creating a network topology for Free-Space Optical (FSO) wireless devices. Initially, all the nodes are disconnected. One by one, nodes are grouped to form clusters based on link state table information. Once the number of clusters is equal to one, a topology has been found. This process then runs continuously, continuously monitoring the state of the network and making repairs as links degrade or fail. Order complexity is not explicitly provided in the documentation.

Summary

This chapter provided the background and literature review necessary to understand the key concepts used in this research. First, a discussion on topology control was presented. Next, flow networks were introduced followed by a brief review of maximum and multi-commodity flow, which serves as foundation for subsequent chapters. Lastly, a look at current and relevant research was presented to ensure the reader's comprehension of the problem. The following chapter describes the formal methodology.

III. Methodology

Chapter Overview

The purpose of Chapter III is to detail the methodology used in this research. The previous chapters have provided a brief introduction and background information for the material necessary to understand the methodology. First, a pedagogical example is given to help illustrate the problem. Next, the knapsack and greedy front-ends are explained. Lastly, a detailed description of the four different methods used to solve the problem examined.

Problem Example

Consider a network with $n = 4$ nodes. Each node i has an arbitrary number interfaces of type f, u_{if}. In a fully connected network, there are

$$E = \left(\frac{n(n-1)}{2} \right) \tag{12}$$

edges[28] where each node is connected to all other nodes. In this example, assume $f = 2$ interface types. Hence, there can be at most n*(n – 1)* or $n^2 - n$ edges (E x f). As illustrated in Figure 10, the example network has 16 - 4 = 12 edges (assuming full duplex).

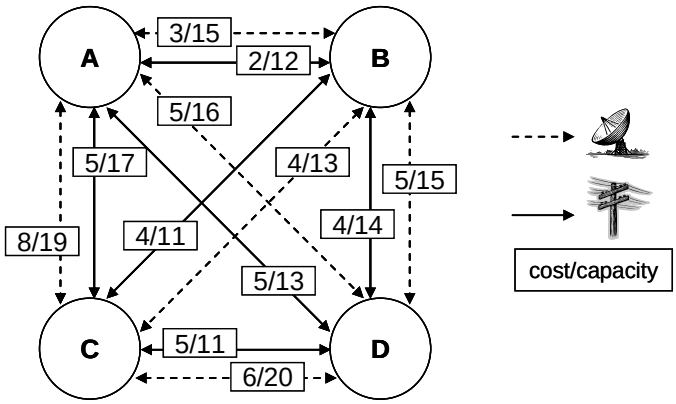

Figure 10. 4-node network with 2 unique interfaces at each node.

As with many networks, each edge has a fixed capacity and an associated cost. These are labeled in Figure 10. For the purpose of simplicity, this example assumes there is a commodity flow requirement for each source/destination pair (this follows Erwin's model described in previous chapter). Thus, with four nodes, there are exactly $K = n^2 - n = 12$ commodities that must flow across the network. For each of these, there is an associated bandwidth required to send across the network. Additionally, a value attribute is randomly assigned. This property helps determine the order in which commodities are solved (i.e., a priority). Furthermore, an edge value vector is given to help identify priorities discussed later. Table 2 lists the commodities for this example.

Table 2. List of commodities for the example network.

Comm (k)	Traffic Source	Traffic Sink	Value	Bandwidth Required
1	a	b	1	3
2	a	c	5	4
3	a	d	5	5
4	b	a	4	3
5	b	c	2	5
6	b	d	2	6
7	c	a	6	4
8	c	b	9	5
9	c	d	9	7
10	d	a	4	5
11	d	b	3	6
12	d	c	9	7
			Total:	60

Notice that this network must have enough capacity to handle at least 60 units of flow, else the problem is not solvable from the beginning. The following table (Table 3) lists the network capacity broken out by edge.

Table 3. Edge list for the example network.

Edge Source	Edge Sink	Interface Type	Fixed Cost	Edge Capacity
a	b	satellite	3	15
a	b	land line	2	12
a	c	satellite	8	19
a	c	land line	5	17
a	d	satellite	5	16
a	d	land line	5	13
b	c	satellite	4	13
b	c	land line	4	11
b	d	satellite	5	15
b	d	land line	4	14
c	d	satellite	6	20
c	d	land line	5	11
		Total:	56	175

This example network can easily handle the traffic requirement presented above. For simplicity and space concerns, the cost per unit flow matrix is omitted. This information is still required to solve the problem as it determines the cost a route a commodity over a particular path.

Erwin's input files were randomly generated in order to prevent a bias within the test sets. Due to the amount of information required to solve the problem (i.e., costs, network information, etc), generating data sets modeling larger networks was memory intensive. To alleviate this problem, a different input format is used, based on the input for HIPR, "an efficient implementation of a push-relabel algorithm for the maximum flow/minimum cut problems[29].". This new format was adopted to provide a means to use unique cost metrics such as link characteristics and commodity preferences.

Because this research is to be compared to Erwin's, however, the exact same input files are still required to be acceptable input. Thus, a conversion routine is implemented to accommodate this. The program is therefore equipped for better comparison of the results of both experiments which is further explained in Chapter IV.

Recall that commodities are determined by the size of the network. Therefore, as the size of the network increases, so does the number of commodities (by a factor of n^2). Consider a relatively small of network of $n = 10$ nodes, implying $K = n^2 - n = 90$ commodities. One of the problems with calculating the solution is determining the set of commodities to use. This can be modeled after the power set, denoted by $\mathcal{P}(S)$, which is defined as the set of all subsets of set S. The cardinality of $\mathcal{P}(S)$ is derived by the following formula: $\mathcal{P}(S) = 2^n$[30]. By this fact, there are 2^{90} (a very large number)

combinations possible. Rather than enumerate each element in the power set using a linear programming model, the problem is mapped to a well-known set of maximum flow algorithms.

As noted in Chapter II, maximum flow algorithms attempt to send as much flow as possible through a flow network from a source to a destination. With topology control, the objective is to maximize throughput by determining a feasible configuration. Moreover, the order of complexity of these algorithms exhibited is a desirable characteristic. Therefore, it is logical to explore the solution potential of this class of algorithms.

The methodology is partitioned into two main phases: selecting a combination of commodities and determining if the selected combination fits in the network. Each phase is now discussed.

Choosing the Combination

Each commodity has a variety of fields to uniquely identify itself within the network. Such fields include an identification (id) number, a source, a sink, a traffic demand, a value (i.e., worth), and a preference vector among others. The preference vector is a list of desired edge characteristics which can be used to model costs in cases where a commodity flow cost is not specified. Using this information, the subset of commodities that provide the greatest benefit to the network is determined. To do this, two steps are taken.

1) Sort the commodities

2) Determine the combinations to try

To accomplish first step, a density function is defined as the ratio between a commodity's value and its demand. That is, the priority of commodity $k = k_{value} / k_{demand}$. This gives preference to commodities that provide the best value for their cost as opposed to sorting only value or only by demand.

Once the commodities are presorted, the next step is to determine the combinations to attempt to solve. It makes sense that solution sets closer to the optimal solution would have increasingly more valuable commodities. Thus, to determine an optimal set of commodities, two approaches are examined.

Knapsack

The first approach is a formulation of the well-known knapsack problem. The knapsack problem arises when you want to maximize the value of a particular set of items while adhering to a strict cost (e.g., weight) constraint. The knapsack problem is formulated as follows[31]:

$$\begin{aligned} &\text{let } W = \text{demand or cost constraint} \\ &\text{let } V = \text{value goal} \\ &\text{let } K = \text{set of commodities} \\ &\forall k \in K,\ \text{find } K' \subseteq K \ni \sum_{k \in K'} k_{demand} \le W \wedge \sum_{k \in K'} k_{value} \ge V \end{aligned} \tag{13}$$

Because the goal is to maximize value, the latter half of the equation above can be modified such that the sum of values for the solution set is optimal. The implementation used in this research comes from the dynamic programming solution to the knapsack problem. This is illustrated in Figure 11.

```
1  for each commodity k in K
2  begin
3     for each weight w in W
4     begin
5        if( k_demand > w ) then
6           A[ k ][ w ] = A[ k – 1 ][ w ]
7        else
8           A[ k ][ w ] = max( A[ k - 1 ][ w ], k_value + A[ k - 1 ][ w – k_demand ] )
9        end-if
10    end-loop
11 end-loop
```

Figure 11. Dynamic Knapsack Pseudocode[32].

This is the generic representation of the knapsack, shown for the purpose of illustrating the process; however, the following details its usage within this research. Let *A* represent a 2-dimensional array of size *K x W*. Cell *A[k][w]* of the knapsack represents a possible combination to try. Each cell is also comprised of the value of the combination solved, the residual graph (i.e., the graph used by the net flow solver), and some Boolean variables flagging the success or failure of an attempted net flow, and a pointer to the previous successful cell. During each step in the algorithm, a decision must be made whether to run the cell's representative combination or not.

Referencing Figure 11, lines 1 – 4 and 10 – 11 ensure each cell in the knapsack is visited. For each cell, there are two possible cases. Case 1 (lines 5 – 6) is invoked if the current commodity's demand is larger than the capacity of the current cell. Otherwise, case 2 (line 8) is invoked. Case 2 is further divided into two sub-cases in which the new commodity *k* adds value or not. If not, the cell is updated with the previous cell's data and the algorithm proceeds to the next cell. If the commodity adds value, then a net flow must be performed to determine if the new combination can be satisfied by the network.

Upon a successful completion, the residual graph is saved and may be used in subsequent net flow runs. Recall that the knapsack's dimensions are *K x W*, thus in the worst case, the net flow routine could be performed *K x W* times. Additionally, the knapsack formulation used is recursive in its definition. This is a crucial time-saving step because it minimizes the amount of flow the net flow routine has to solve. Without it, every net flow run would have to try and solve the entire combination. Once the knapsack runs to completion, cell *A[K][W]* of the knapsack array contains the final solution possible.

Greedy Technique

The second approach to choosing a combination is the greedy technique. As the name implies, this technique always chooses the best option available at any given time and the choice made is irrevocable[33]. Recall from Chapter II that the greedy method for maximum flow algorithms suffer from getting "stuck" in a local optimum during the search process. This is because no global information is used to make decisions. On the other hand, greedy techniques are often acceptable substitutes for approximation algorithms as they exhibit much faster run times than other, more exhaustive search techniques.

The greedy approach taken for this research starts with selecting the best commodity from the list of commodities that have yet to be explored. Since this list is already sorted (mentioned above), making the greedy decision is simplified to checking the next commodity in the list. Let K^* be the list of sorted commodities. For each k^* in K^*, if k^* is solvable in the current residual flow network, it is added to the "solved" list and the resulting residual graph is saved for the next iteration. If k^* can not be solved, the

commodity and the residual graph are discarded. Once all commodities have been explored, the solution is stored in the "solved" list.

Solving the Combination

The second phase is net flow computation. This phase determines whether a particular flow/commodity or a set of commodities can be satisfied by the network. Briefly stated in Chapter II, there are a number of maximum flow algorithms available, however this research focuses on the Edmonds-Karp and Pre-flow Push algorithms. First, a brief discussion on the algorithms' objectives in general are provided, followed by a more in-depth presentation explaining how they compute solutions.

Each algorithm, which has been modified to operate both with fixed edges as well as a set of potential edges, requires the same information to solve a particular flow. Foremost is the graph (i.e., the network). The graphical representation of the networks used in this research is not complex. A network object is instantiated from the parent class. Each network object contains a list of nodes and a list of commodities. Each node then contains a list of regular edges and potential edges. Additionally, the net flow algorithms require the list of commodities. The final piece of information required is the heuristic which the algorithm should use. The heuristic options are a breadth-first search (BFS) or a best-first search (BestFS). The following subsections detail the algorithms and heuristics in more detail.

Edmonds-Karp

The Edmonds-Karp Maximum Flow algorithm (also known as the Labeling Algorithm[24]) is a variant of the Ford-Fulkerson Maximum Flow algorithm[21]. The

Ford-Fulkerson method is straightforward: find a path from the source to the sink (called augmenting paths), push flow along that path, update edge capacities and repeat until no further augmenting paths are available. The primary difference of the Edmonds-Karp method from the Ford-Fulkerson method is how augmenting paths are found. Ford-Fulkerson uses depth-first search (DFS) approach to finding the sink node, whereas Edmonds-Karp utilizes a BFS. The result is a faster runtime because augmenting paths are encountered in order of the number of hops from the source to the sink (i.e., shortest path first)[24]. Before introducing the implementation specifics, the pseudocode for the general Edmonds-Karp maximum flow algorithm is given (Figure 12).

```
1  Edmonds-Karp()
2     label sink
3     while( sink is labeled )
4       un-label all nodes
5       initialize predecessor for each node to 0
6       label the source and add source to a List
7       while( the List is not empty )
8          i = remove a node from front of List
9          for each existing edge i to j
10            if node j is unlabeled
11               label j and set predecessor( j ) = i
12               add j to List
13            end-if
14         end-loop
15      end-loop
16      if sink is labeled
17         P = augmenting path via predecessor labels
18         b = minimum capacity of edges in P
18         add b units of flow to each edge
20         update residual graph
21      end-if
22   end-loop
23 End
```

Figure 12. General Edmonds-Karp Maximum Flow Algorithm[24].

In the algorithm, labeling a node implies that node has been visited. Line 2 is an initialization step required to enter the subsequent loop. The `while` loop on lines 3 – 22 executes until no further augmenting paths exist. Once the loop is entered, lines 4 – 6 are executed as a precursor to the inner `while` loop on lines 7 – 15. This loop attempts to find an augmenting path. During this process, predecessor information is preserved in order to identify the path (`List` only contains the nodes to be explored). The next step is to increase flow along the augmenting path. Obviously, this step (Lines 16 – 21) is only executed if an augmenting path is found.

To accommodate the use of potential edges, a Boolean flag is maintained. The first attempt to solve a commodity always tries to use any fixed edges that are already in place. Ideally, if traffic requirements can be fulfilled with links in which the cost is already sunk (i.e., fixed edges), the algorithm should try to do so. Only when this fails are potential edges considered. This requires minimal modification to the algorithm in Figure 12. It is accomplished by changing line 9 to read "for each existing edge or potential edge from *i* to *j*." Note that this may result in less optimal results, in terms of the match between the desired links characteristics and a commodity's preference vector, but it should help to maximize the number of admitted commodities in the network.

Another requirement is to modify the algorithm for multiple commodities rather than the maximum flow. This requires only a few modifications as well. By nesting lines 2 – 22 within a `for` loop, the algorithm can be applied for as many commodities as necessary (i.e., for each commodity *k* in *K*). However, rather than the maximum flow,

only the demand of the commodity needs solving. Therefore, safeguards are added to assure the proper amount of flow is augmented on the edges.

Essentially, the implementation above describes the first Edmonds-Karp method that is used. The second method involves a slight modification on how the edges are chosen for augmenting paths. Rather than using a BFS when searching for an augmenting path, a BestFS is implemented. This heuristic sorts the choice of edges according to the variable cost of routing the particular commodity over the edge. To implement this, a priority edge queue is utilized. This transforms the search from breadth-first into a best-first search which analyzes the next best edge at each step.

The implementation modifications to Figure 12 are reflected in an updated version of the pseudo-code in Figure 13.

```
1  Edmonds-Karp()
2     for each commodity k in K
3        label sink( k )
4        while( sink( k ) is labeled and demand-left( k ) != 0 )
5           un-label all nodes
6           initialize predecessor for each node to 0
7           label the source( k ) and add to a List
8           while( the List is not empty )
9              i = remove a node from front of List
10             for each existing edge i to j
11                if node j is unlabeled
12                   label j and set predecessor( j ) = i
13                   add j to List
14                end-if
15             end-loop
16          end-loop
17          if sink( k ) is labeled
18             P = augmenting path via predecessor labels
19             b = min( minimum capacity of edges in P, demand-left( k ) )
20             add b units of flow to each edge
21             update residual graph
22          end-if
23       end-loop
24    end-loop
25 End
```

Figure 13. Updated Edmonds-Karp Algorithm.

Pre-flow Push

There are a number of variations for the Pre-flow Push algorithm as noted in Chapter II. The maximum flow algorithm described here, however, is defined by Lewis and Deneberg[23]. Before introducing the implementation specifics, the pseudocode for the general algorithm is given (Figure 14).

```
1  Pre-Flow Push()
2     initialize all flows to 0
3     initialize value to 0
4     while( true )
5        A = BuildAugmentingNetwork( G )
6        ComputeLayers( A )
7        if( sink was not layered )
8           return value
9        else
10          PruneAugmentingNetwork( A )
11          CalculateVertexCapacities( A )
12          while( sink has unsaturated incoming edges in A )
13             v = FindLeastCapacityVertex( A )
14             value += Capacity( v )
15             PushAndPullFlow( A, v )
16          end-loop
17       end-if
18    end-loop
19 End
```

Figure 14. General Pre-flow Push Max Flow Algorithm[23].

Lines 2 – 3 are initialization steps. The outer `while` loop (lines 4 – 18) is instantiated as an infinite loop. It provides a similar check to the outer `while` loop in the Edmonds-Karp algorithm in that each iteration is an attempt to find a path from the source to the sink.

Line 5 builds what the authors call an augmenting (or scratch) network. The resultant network (*A*) is a copy of the original network (*G*) such that any edges that have been saturated (i.e., have no residual or excess capacity available due to previous iterations) have been removed. It is not a required step, but does offer some optimization that saves time when exploring nodes further in the algorithm.

Next, each node's layer is computed (line 6). A node's layer is defined as the distance from that node to the sink. This technique proves useful and essential in

determining if the sink is reachable and it also identifies which nodes are on a direct path from the source to the sink. For example, the nodes on a direct (shortest) path would have layers that are one plus its predecessor. If the sink does not get layered on a particular iteration, then the algorithm terminates because no source-sink paths exists. Otherwise, it continues on to line 10.

Once node layers are computed, the augmenting network is "pruned" (line 10). This process removes nodes and constituent edges that are not part of a source-sink path of shortest length (identified by the layer of the sink), hence pruning. This step reduces the work required further on as did building the augmenting network.

Rather than augmenting flow from the source to the sink like the Edmonds-Karp algorithm discussed previously, the Pre-Flow Push algorithm uses a combination of push and pull techniques. First, each node's capacity is calculated (line 11) by taking the minimum of the sum of the incoming edge capacities and the sum of the outgoing edge capacities—a node can only receive as much flow as its edges can handle. This step makes finding the node with the smallest capacity very easy. Next, a node v with the lowest capacity, denoted by c, is selected as the starting point (line13). A series of push and pull operations are then recursively called until the sink has received c units of flow and the source has pushed c units of flow. In other words, node v pushes c units of flow out on its outgoing edges and pulls c units of flow in from its incoming edges. This saturates node v. Next, any node that had flow pushed to or pulled from must perform the same push/pull technique until c units of flow is pulled from the source and pushed to the sink. Finally, the augmenting network is updated by removing the nodes and edges

that are saturated. This process (inner while loop, lines 13 – 15) continues until all incoming edges of the sink are saturated. At this time, all source-sink paths of the shortest length have been fully utilized, and the algorithm can begin looking at paths of the next shortest length.

It is important to note that by selecting the vertex with the least capacity as a starting point to augment flow through the network, the algorithm ensures that any other node on the augmenting path will be able to allocate enough capacity among its edges to successfully push/pull at least *c* units of flow[23].

The Pre-flow Push algorithms must also be modified to accommodate potential edges and multiple commodities. These changes are nearly synonymous with Edmonds-Karp, thus a detailed explication is omitted. The two variants are now discussed.

The first approach is based on the Pre-Flow Push algorithm outlined above. First, the maximum flow between the source and the sink is attempted. If the demand cannot be met given the preliminary network, potential edges are activated (i.e., permitted to be explored). Potential Edge choices are made separately as opposed to "on the fly" like the Edmonds-Karp algorithms. The heuristic uses BFS starting at the node with the highest priority and has potential edges available for consideration. If an edge is deemed useful, it is locked in place, becoming part of the original network. After making the potential edge choices, the new topology is analyzed again to determine if more flow can be allocated to solve the commodity in question.

The second approach uses a slightly different method for finding paths. Utilizing the same techniques as the previous approach, this heuristic determines if a demand can

be met without potential edges. Potential edges are activated in an effort to add additional capacity to the network. When making decisions about which potential edges to explore, the algorithm will sort the edges based off the variable cost to route a particular commodity over the edge in question.

The implementation modifications to Figure 14 are reflected in an updated version of the pseudo-code in Figure 15.

```
1  Pre-Flow Push()
2     for each commodity k in K
3       initialize all flows to 0
4       initialize value to 0
5       while( true )
6          A = BuildAugmentingNetwork( G )
7          ComputeLayers( A )
8          if( sink( k ) was not layered )
9             if( potential edges have not been activated )
10               activate potential edges
11            else
12               return failure
13            end-if
14         else
15            PruneAugmentingNetwork( A )
16            CalculateVertexCapacities( A )
17            while( sink( k ) has unsaturated incoming edges in A )
18               v = FindLeastCapacityVertex( A )
19               value += min( Capacity( v ), demand-left( k ) )
20               PushAndPullFlow( A, v )
21               demand-left( k ) -= value
22               if( demand-left( k ) == 0 )
23                  go to next commodity
24               end-if
25            end-loop
26         end-if
27      end-loop
28   end-loop
29 End
```

Figure 15. Updated Pre-flow Push Algorithm.

Potential Edges

By definition, a potential edge could possibly connect to multiple end points. However, only one connection is permitted at a time. To represent a single potential edge, each possible connection is split into a separate pseudo-edge. Each pseudo-edge (it is not an actual edge unless it is used) is then assigned a group id number. Only one pseudo-edge per group is permitted to be in use. Consider the following example.

Figure 16 depicts a simple flow network in which node 2 has one potential edge (two pseudo-edges). The potential edge can either point to node 3 or node 4, but not both. The choice in this example is quite simple. If pseudo-edge 2 → 4 is utilized, the sink will be unable to receive flow, thus the obvious choice is 2 → 3.

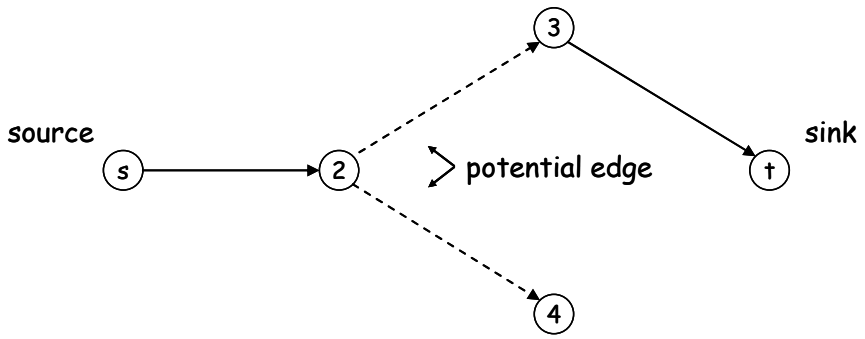

Figure 16. Potential edge illustration.

Summary

This chapter started by providing a necessary example to illustrate the problem at hand. Then, the knapsack and greedy front-end methods used to determine the commodity combination was explained. Lastly and the core of the research, the Edmonds-Karp and Pre-flow Push maximum flow algorithms were detailed as the basis for determining if a given commodity combination could be satisfied by the network. Furthermore, modifications for accommodating potential edges and multiple commodities were examined. Given the two front-end methods, which can each utilize one of four maximum flow methods, there are a total of eight unique approaches to solving the topology control problem (See Figure 17).

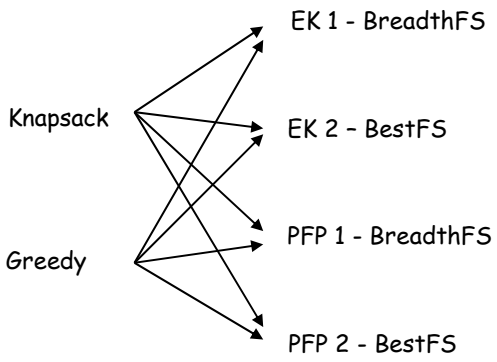

Figure 17. Methods for solving the problem.

The next chapter details the experimental design and analytical results gathered through tests on this methodology.

IV. Analysis and Results

Chapter Overview

This chapter provides a detailed description of the experiments used to test the methodology described in Chapter III. There are two primary modes of experiments employed: the comparison of the results of Erwin's tests[13] with that of this research, and sole evaluation of the unique methods themselves with respect to one another. This chapter details the setup and design of such experiments. The results are then presented, analyzed, and critiqued. Ultimately, conclusions are drawn upon and hypotheses are readdressed to highlight the salient features of the research.

Design of Experiments

The motivation from which this research is derived stems from the baseline results provided by Erwin[13]. His approach was aimed at minimizing the cost of the network. This research abstracts the cost of edges and routed flow away such that the goal is more geared toward optimizing commodities within the network. That is, the characteristics of the traffic in the network are more important than the final cost. In order to ensure proper comparison and some guarantee that the data is meaningful, a post-processing phase was implemented to capture the same metrics presented by Erwin. The time required to collect this data was not counted toward the overall computation time of the methods.

The metrics recorded and used to compare and contrast against Erwin's results are presented in Table 4:

Table 4. Metrics used for comparison with Erwin's results.

Metric	Description
Link Cost	The cost per use a particular link in the network (without regard to the traffic flowing over the link)
Flow Cost	The cost to route a given flow across a particular link
Total Cost	The sum of the Link Cost and the Flow Cost
Number of Hops	The average number of hops the commodities within the network exhibit from their respective source and sink nodes
Diameter	The largest realized distance between any two nodes of the network
Dropped Commodities	The number of dropped commodities the network experienced
Run time	The total time required to solve the problem

To calculate the link, flow and total cost of the solution network, the cost matrices provided in Erwin's input files were read into memory and utilized in the post-processing phase. While the remaining metrics were also calculated after the solution was found, no additional information was required apart from the solution network itself.

Initial experiments were modeled directly after Erwin's. That is, input files were created with random characteristics for networks of size 10, 15, 20, 25, 30, 35, and 39 nodes. Due to memory insufficiencies and the input file format, networks of 39 nodes or more were not examined. To remain consistent, however, the exact same input files were used. Each input file contained the number of nodes, a list of commodities, a node incidence matrix, and various cost matrices for determining goodness. Recall that there are eight total methods used to solve the problem. For each input file, 10 trials were run per method (i.e., 80 trials per input file). Results in the form of the metrics previously described were output to individual text files. Data collected from the files were inserted into a spreadsheet where the averages could be extracted, ensuring the data is normalized to account for instances that strayed from the standard deviation.

Test Computer Specifications

All experiments were completed on a Gateway desktop personal computer running Fedora Core release 4 from Red Hat Enterprises. The machine boasts an Intel Pentium 4 processor running at 3.40 GHz with 2 MB cache. It also has 4 GB of virtual memory, 3.6 GB of which is available to the user. All code was written in C/C++ and compiled and tested using the GNU Compiler Collection, aka GCC, version 4.0.0.

Order Complexity

Using the knapsack method to determine which combinations of commodities to run in a net flow, the order complexity can be computed as $O([\text{net flow}] \times K \times W)$. For the remainder of this document, let $O_{EK}(X)$ denote the order complexity of the Edmonds-Karp algorithm and $O_{PFP}(X)$ denote the order complexity of the Pre-flow Push algorithm. Recall that that the Edmonds-Karp and Pre-flow Push maximum flow algorithms exhibit an order complexity of $O_{EK}(V \times E^2)$ and $O_{PFP}(V^3)$, respectively. Thus, in the (best) case where only a single net flow is run, the knapsack method complexity is $O_{EK}(V \times E^2)$ or $O_{PFP}(V^3)$. On the other hand, the worst case is illustrated by running a net flow in each of the $K \times W$ cells of the knapsack. Recall that K is the number of commodities and W is the maximum weight of the knapsack. Since $K = n^2 - n = V^2 - V \sim O(V^2)$, the order complexity can be further simplified from $O_{EK}(V \times E^2 \times K \times W)$ to $O_{EK}(V^3 \times E^2 \times W)$ and from $O_{PFP}(V^3 \times K \times W)$ to $O_{PFP}(V^5 \times W)$. Neither the set of edges, E, nor the maximum weight, W, are necessarily dependent on the number of nodes, however, they can both be sufficiently larger—especially for dense networks where a fully-connected graph has $O(V^2)$ edges and networks with large capacity edges which increases the size of the

knapsack. Thus, it is difficult to establish an order complexity completely in terms of n or V. Even if the graph is very sparse, such as a spanning tree, the complexity is still suitably poor at $O_{PFP}(V^5 \; x \; W)$ at best. The results discussed momentarily the poor execution time performance expected given its complexity analysis.

Using the greedy method for choosing commodities, the outlook on complexity is not quite as unenthusiastic. Recall that the greedy method tries to solve each commodity one-by-one by passing the net flow routine the previous commodity's residual graph. Therefore, for each of the K commodities, a net flow routine is called only once. Depending on which method is chosen the order complexity can be as low as $O_{PFP}\ (V^3 \; x \; K)$ for Pre-flow Push and as high as $O_{EK}\ (V \; x \; E^2 \; x \; K)$ for Edmonds-Karp. By simplifying the V and K variables into one term, the order complexity can be roughly equated to $O_{PFP}(V^5)$ and $O_{EK}\ (V^3 \; x \; E^2)$. For very dense graphs, aka a fully-connected graph, the upper bound for the Edmonds-Karp heuristics could be as high as $V^3 \; x \; E^2 = V^3 \; x \; V^2 \; x \; V^2 = V^7 = O_{EK}\ (V^7)$.

Limitations

Just by examining the run time complexity, it is easy to see that Erwin's assumption (from Ajuha, et al.[24]) of having a commodity for every source/destination pair possible in the network severely hinders the overall performance of the algorithm. It could be argued that this is not particularly illustrative of the real world. Perhaps a more realistic approach would be to model the number of commodities randomly or as a factor of exactly n, rather then n^2. This would reduce the complexity by a factor of n without

taking value away from the problem. Figure 18 illustrates the difference a single factor of *n* can make.

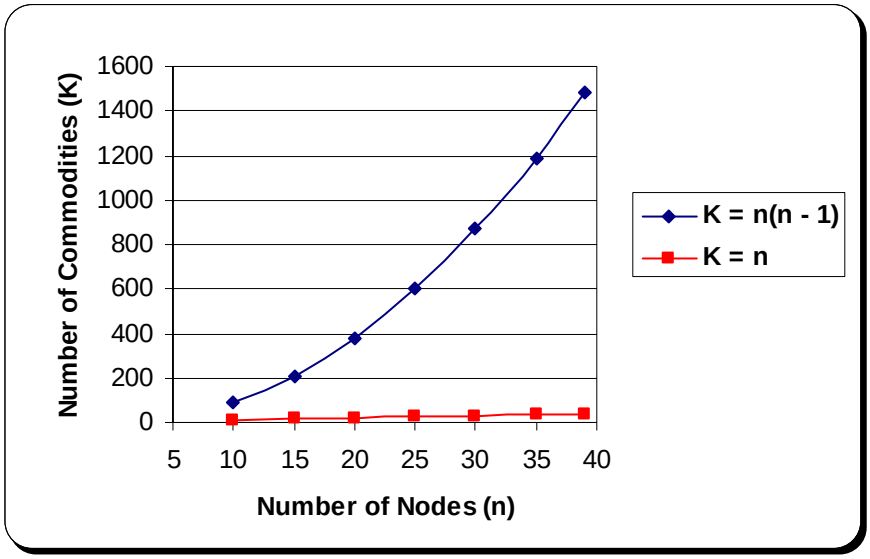

Figure 18. The difference between *n* and n^2 commodities.

Another significant limitation is the availability of random-access memory (RAM). This is especially apparent for tests using the knapsack method. Recall that in the worst case, a net flow can be performed in each of the *K x W* cells in the knapsack. Thus, in order to minimize the amount of computation for each net flow, residual networks from subsequent net flow attempts are saved. Whenever a new net flow attempt is necessary, the residual graph is passed in to the net flow solver. Thus, rather than solving for the entire combination, each net flow attempt is only required to solve a single commodity on the residual network. This optimization, however, proves costly because the data structures used to identify a network and its constituents is memory

intensive. To compromise, a limit is set on the number of graphs that can be saved. Whenever new graphs are created, older graphs are removed to make space. For small problem sizes, this is acceptable and only minor impact is noticeable. However, as the knapsack churns through the *K x W* array, the number of references to previous cells increases significantly. This is problematic as older graphs are replaced by newer graphs (new graphs are more valuable due to the likelihood of getting reused in the near future). The next section highlights the implications of this obstacle.

Results of Experiments

This section provides the results of the experiments for both the knapsack and greedy approaches to include the four net flow heuristics as compared to Erwin's MILP and heuristic methods. To facilitate an easier means of comparing the data, results for all methods are presented in tandem for each of the input files tested (i.e., 10 nodes, 15, nodes, 20 nodes, etc.). This helps build on the comparison between the methods. Following the results by input file are the overall results for each method and input file. This provides insight on scalability and on how the methods perform over time.

The actual input files are much too big to include in this document, however, they are provided electronically if desired. Furthermore, the actual data points from which the charts below are generated are located in Table 5 – Table 10 in Appendix A.

Metric 1: Run time

The first metric analyzed is the run time performance. Run time is one the most important metrics collected. Real world implementations of a topological solver will have to be fast among all other traits as networks are becoming increasingly more mobile

and ad hoc. The following figures (Figure 19 – Figure 23) show the run time performance of each method over the span of the tested networks. Further analysis is provided afterward.

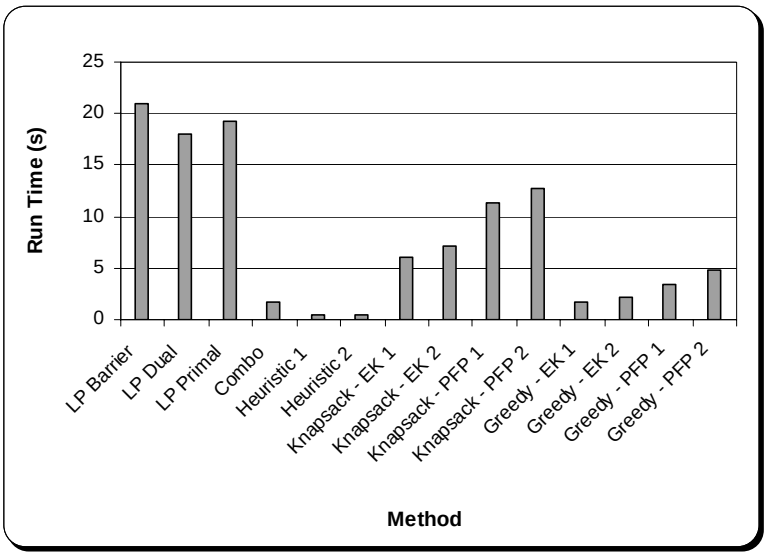

Figure 19. Run time (s) for each method on a 10-node network.

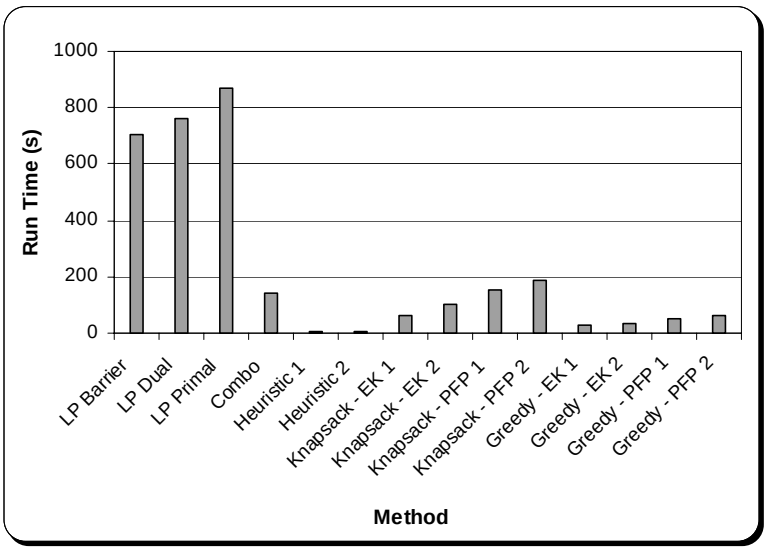

Figure 20. Run time (s) for each method on a 15-node network.

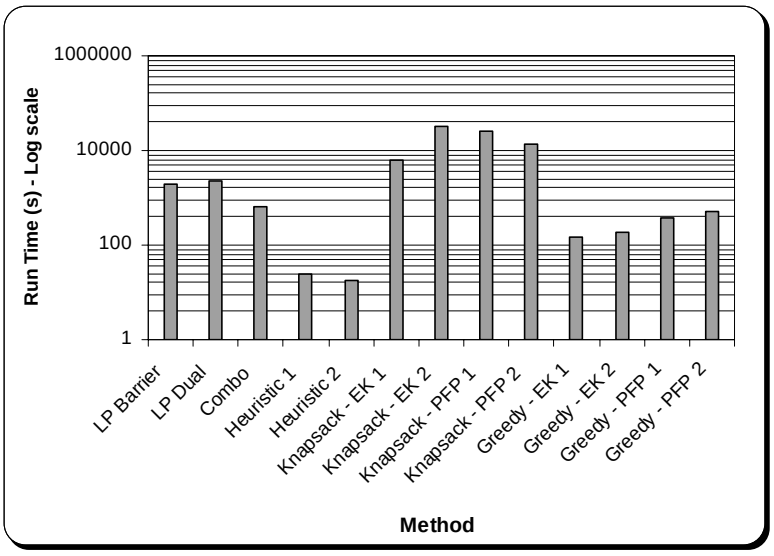

Figure 21. Run time (s) for each method on a 20-node network (logarithmic scale).

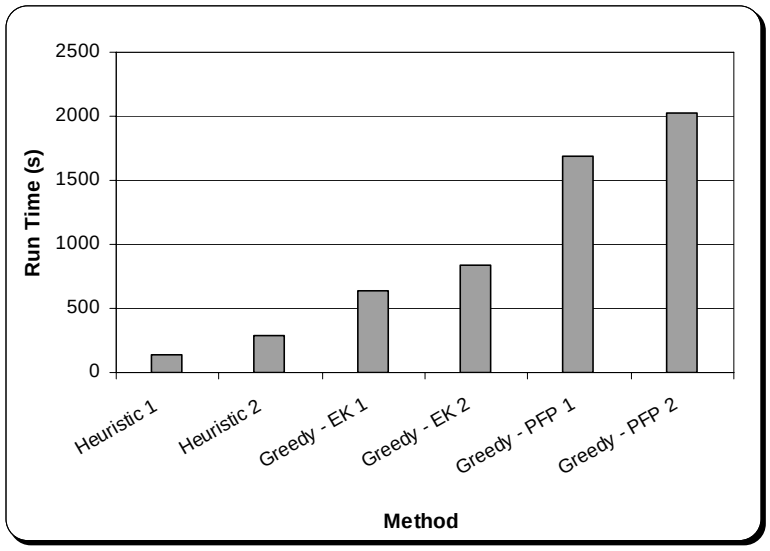

Figure 22. Run time (s) for each method on a 25-node network.

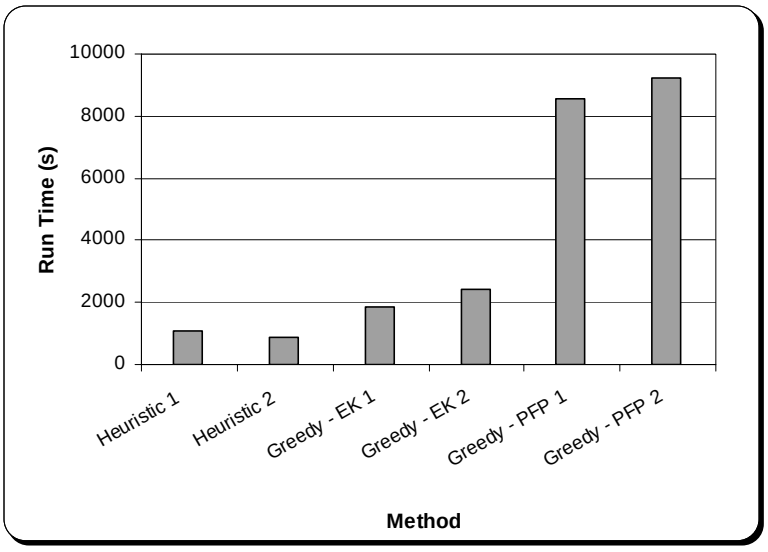

Figure 23. Run time (s) for each method on a 30-node network.

There are a couple of trends clearly evident among these charts. First, while Erwin's LP methods exhibit the highest run times in the experiment, it is also his heuristics that have the slight runtime advantage over the greedy and knapsack methods. However, with the 20-node network, the knapsack method explodes, forcing the use of the logarithmic scale in Figure 21 for comparison (recall the memory limitations previously described). In fact, run time performance is so bad that tests on subsequent networks of larger size were not feasible. This is similar in regards to the LP methods in which data was not provided for larger networks as well.

Also note that the Edmonds-Karp heuristics generally outperform the Pre-Flow Push heuristics often by nearly a factor of two. This is not expected because the theoretical order complexity described above suggests the opposite. There are two possible explanations. First, because the networks are relatively sparse, especially in the early stages of the process when edges just being added, fewer edges are expounded

throughout the search process, thus making the E^2 variable in $O_{EK}(V \times E^2)$ much less of a factor. Secondly, the Pre-flow Push algorithm did not easily lend itself to accommodating potential edges. In fact, a separate routine is used to add potential edges when the existing edges in the network failed. Arbitrarily adding potential edges does not provide an effective means solving flows. Rather, edges must be picked that are useful. Because there is not necessarily a notion of augmenting paths in the Pre-flow Push algorithm, each time potential edges are activated, a separate search is required to identify useful edges. This adds additional time to the overall process.

To get a better picture of the run time performance as whole, the above results are merged into a new chart, Figure 24, which helps illustrate a conglomeration of the results.

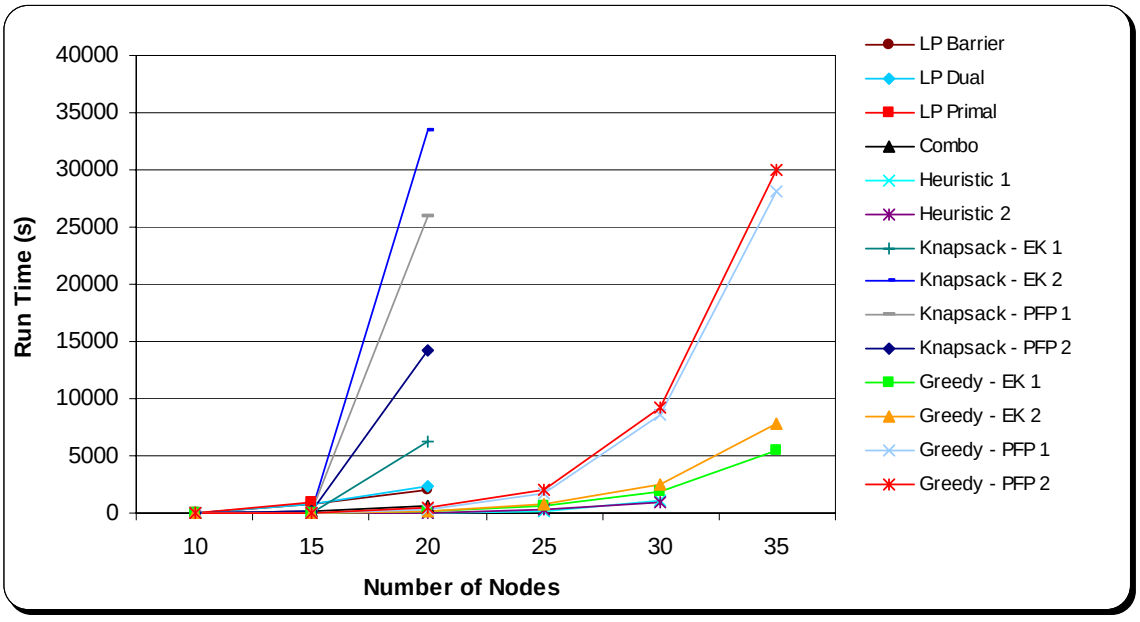

Figure 24. Overall run time performance for all tested networks.

Metric 2: Number of hops

The next metric collected is the average number of hops per commodity. This metric is important as it is associated with delay, a common network metric used in routing problems. To remain consistent, the same approach to introducing the results is used for each metric. The following figures (Figure 25 – Figure 29) present the average number of hops per commodity for each method and for each of the tested networks.

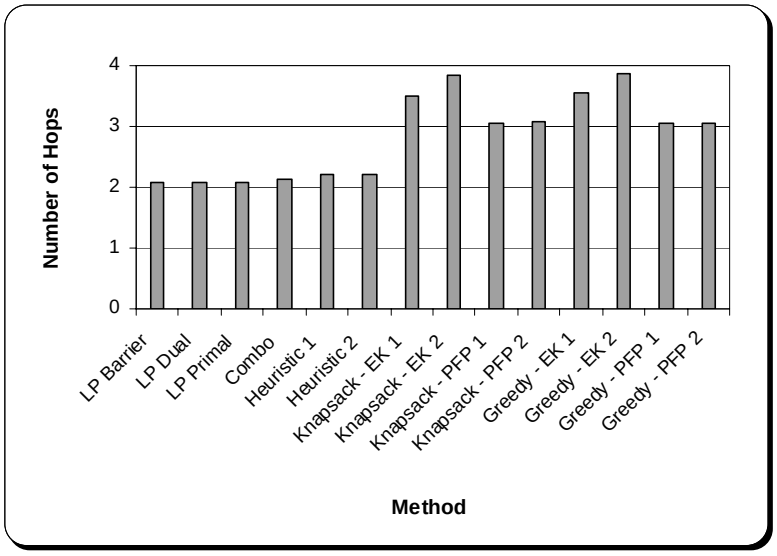

Figure 25. Average number of hops for each method on a 10-node network.

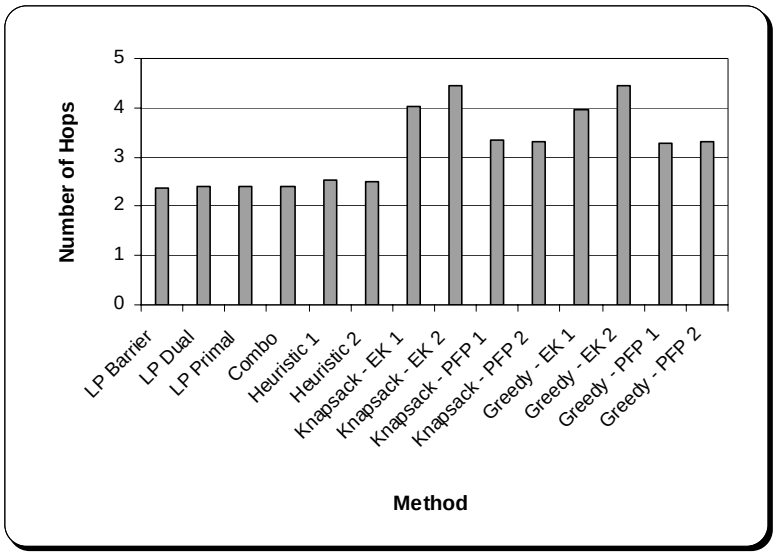

Figure 26. Average number of hops for each method on a 15-node network.

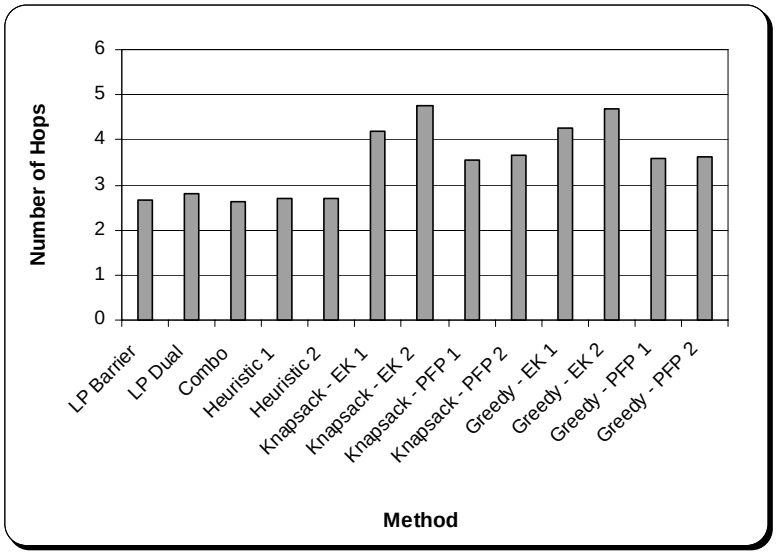

Figure 27. Average number of hops for each method on a 20-node network.

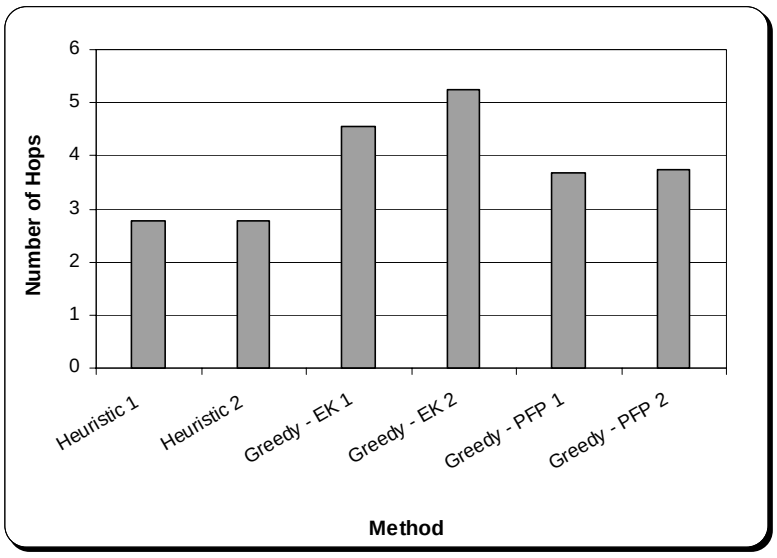

Figure 28. Average number of hops for each method on a 25-node network.

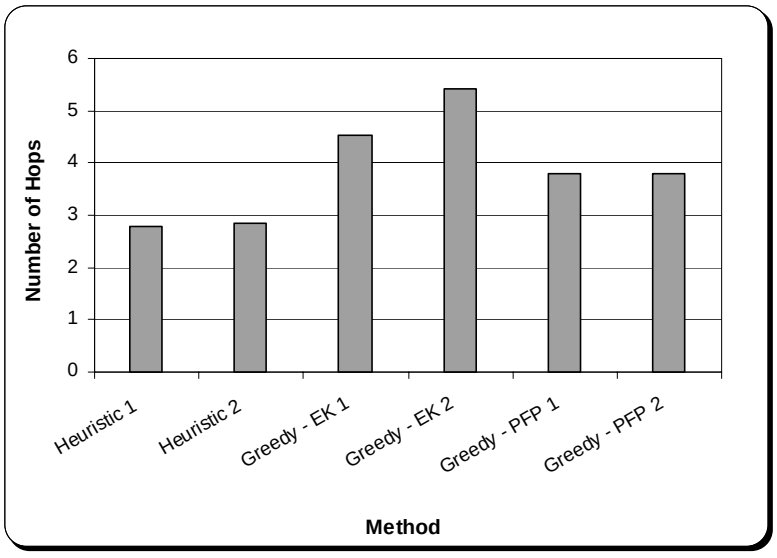

Figure 29. Average number of hops for each method on a 30-node network.

In general, Erwin's LP methods and heuristics have very similar results, hovering around two and three hops per commodity. Similarly, the greedy and knapsack methods

perform on par with one another. However, at best, the results are approximately 50% poorer on average, as produced by the Pre-flow Push heuristics. The Edmonds-Karp heuristics, on the other hand, exhibit somewhat less desirable results ranging from three and half to six average hops per commodity.

On a positive note, the average number of hops appears to scale quite well overall. Referencing the figure below (Figure 30), as the number of nodes increase, the average number of hops seemingly follows a logarithmic curve, albeit because there is at least one instance in which the average number decreases (from 25 to 30-node network, using the Greedy - EK 1 approach), it cannot be completely logarithmic. Nonetheless, its curve is rather appealing and desirable for scalable networks.

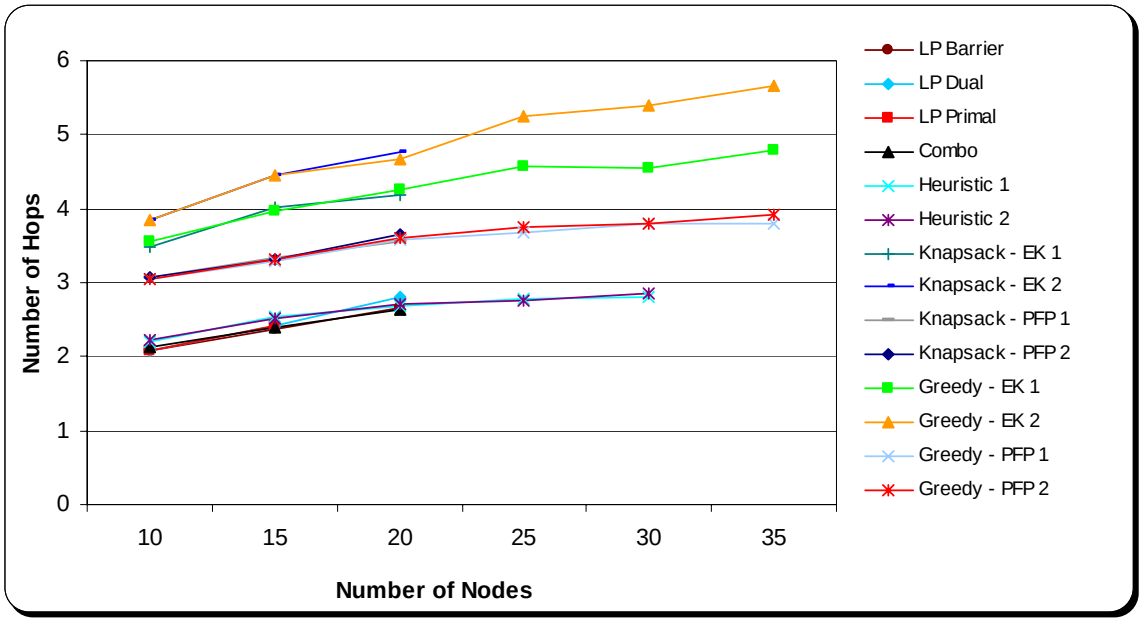

Figure 30. Overall average number of hops for all tested networks.

Metric 3: Dropped commodities

The third metric collected is the average number of dropped commodities. The dropped commodities metric indicates how effective the network is. Essentially, it is equivalent to network throughput—the amount of data that can be in flow at any one time. The results for the number of dropped commodities are introduced in Figure 31 – Figure 35.

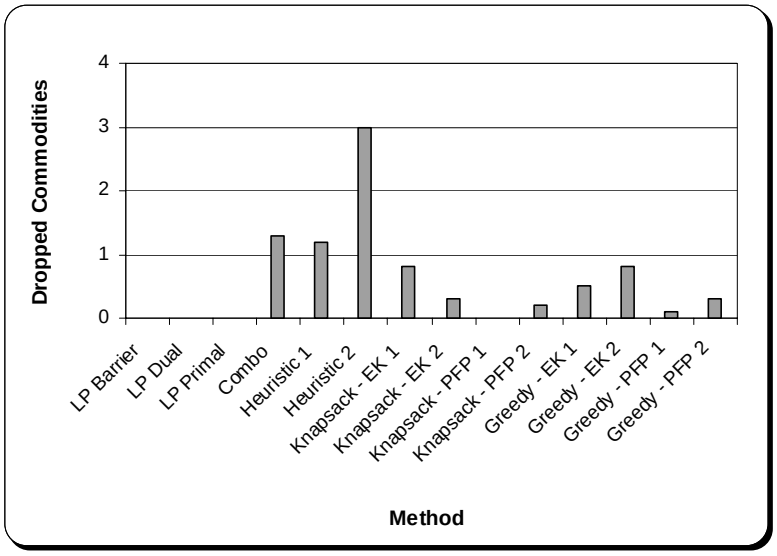

Figure 31. Number of dropped commodities for each method on a 10-node network (total possible = 90).

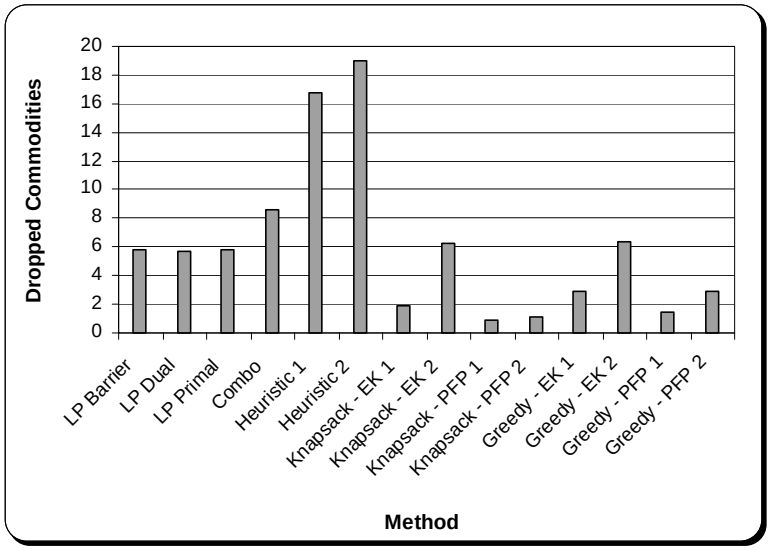

Figure 32. Number of dropped commodities for each method on a 15-node network (total possible = 210).

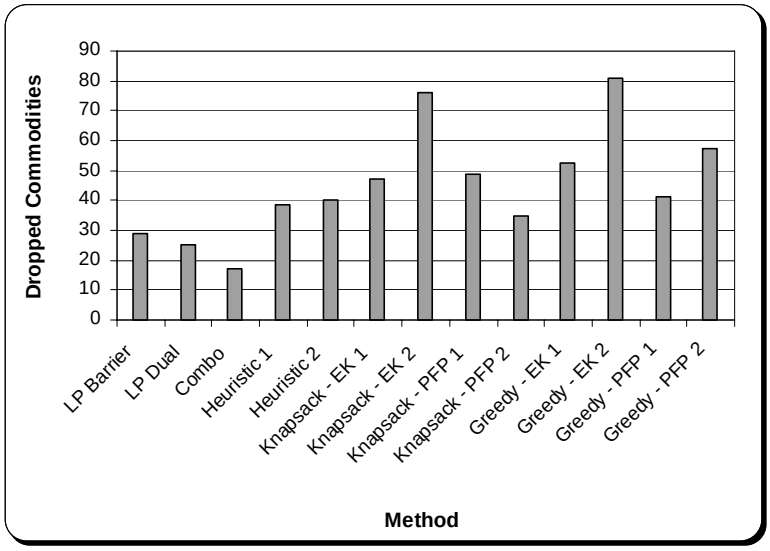

Figure 33. Number of dropped commodities for each method on a 20-node network (total possible = 380).

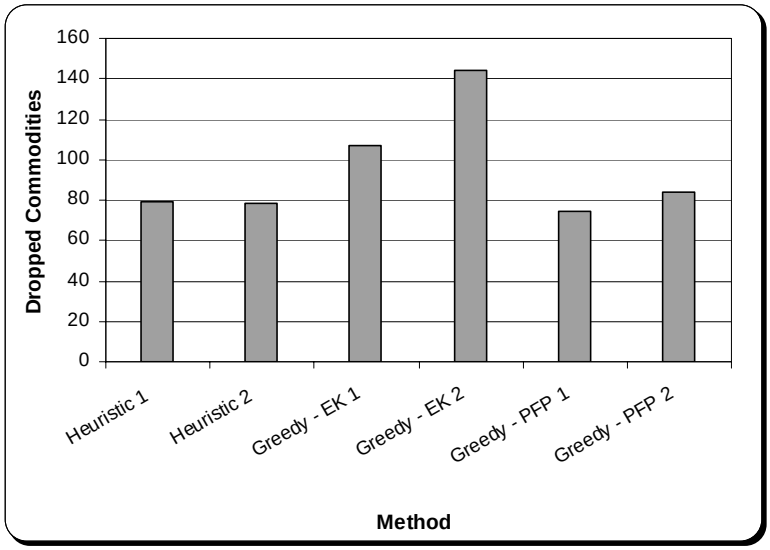

Figure 34. Number of dropped commodities for each method on a 25-node network (total possible = 600).

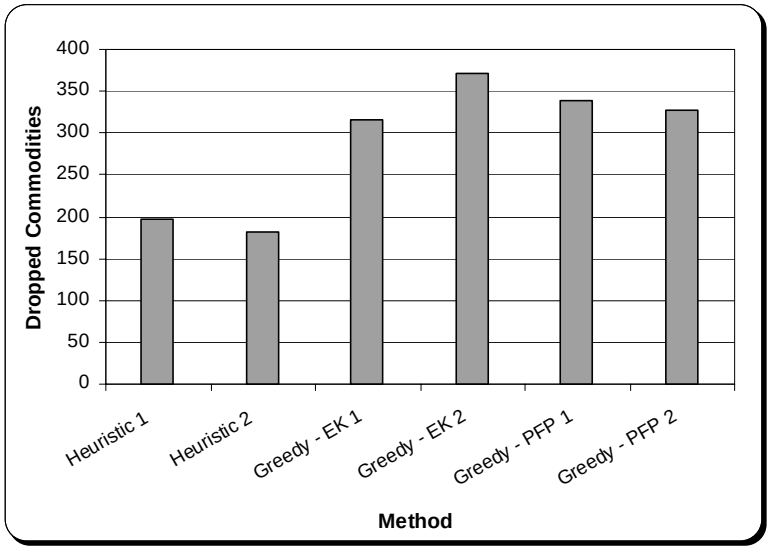

Figure 35. Number of dropped commodities for each method on a 30-node network (total possible = 870).

For smaller network sizes, both the greedy and knapsack methods provide stellar results, especially compared to Erwin's heuristics. In fact, the 15-node network, the

greedy and knapsack methods were able to outperform Erwin's LP techniques. These results suggest the LP model doesn't provide the optimal solution. However, recall that Erwin used cost as his primary objective. Thus, it is possible that the least cost solution solves fewer commodities, primarily because additional commodities could conceivably add significant cost to the solution. Unfortunately, the greedy and knapsack methods do not outperform Erwin's techniques across the board as seen with the larger networks.

Furthermore, supporting the assumption that the number of commodities is of a factor larger than n, then as the network increases in size, the number of dropped commodities is expected to increase significantly as well (recall the limitations of factors of commodities previously discussed).

Figure 36 depicts the overall performance for each network. At first glance, the rise in the number of dropped commodities appears to be relatively minimal. However, larger increases are evident with 30-node networks. With the addition of the 100% trend line, it is apparent that the number of dropped commodities for using greedy heuristics is increasing at nearly the proportion of n^2.

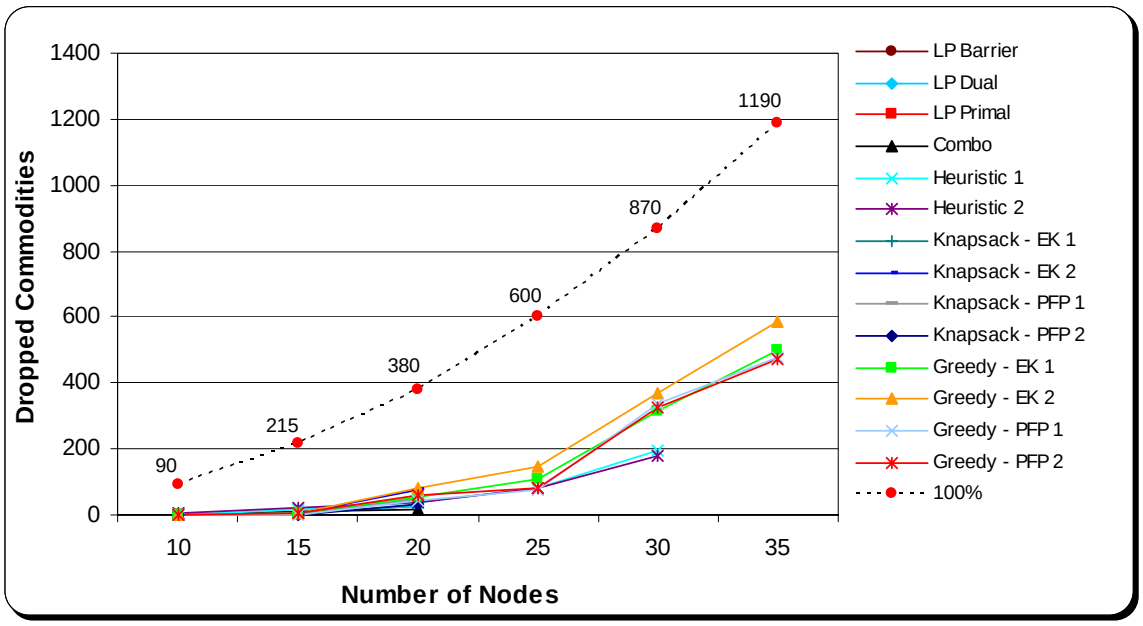

Figure 36. Overall average number of dropped commodities for all tested networks.

Metric 4: Total Cost

The next metric collected is the total cost for constructing the network. As outlined in Table 4, the total cost is the sum of the cost of constructing the links in the network and the cost to route the solved commodities across those links. This metric defines the objective function for Erwin's MILP formulation. Thus, since the MILP formulation provides a cost-optimal solution, neither the greedy nor the knapsack heuristics are expected to outperform the LP techniques. Figure 37 – Figure 41 illustrate the total cost for each method and network size.

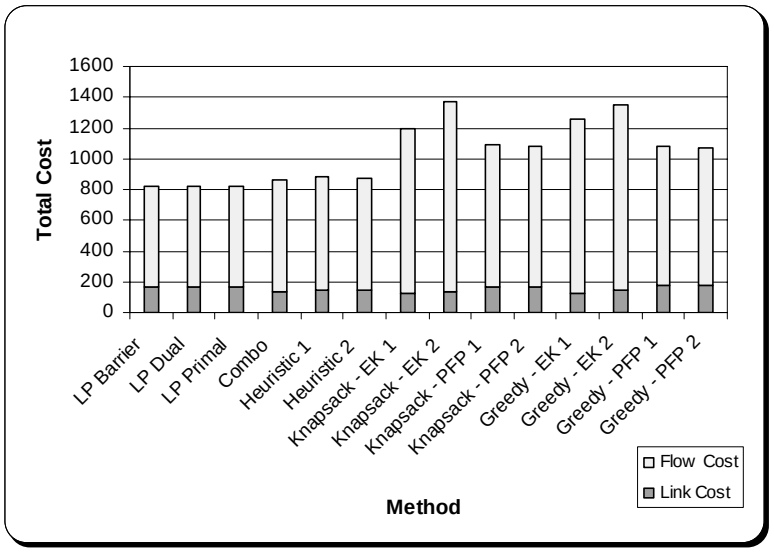

Figure 37. Total cost broken out by link and fixed cost for each method on a 10-node network.

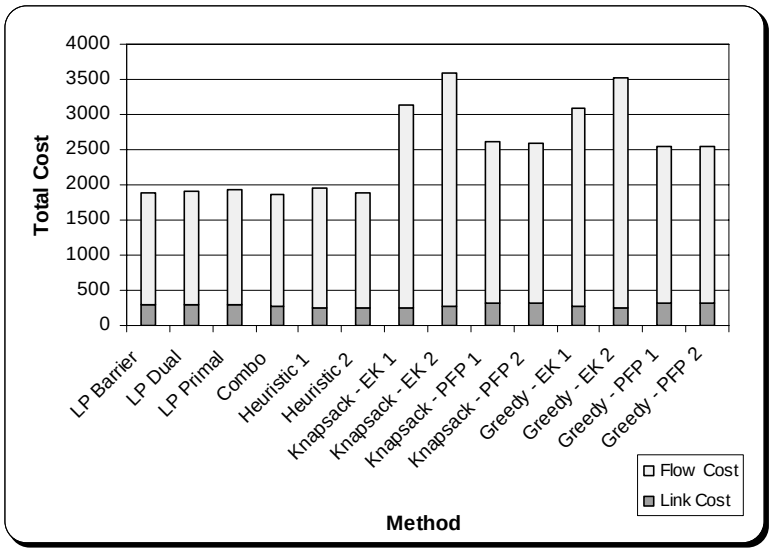

Figure 38. Total cost broken out by link and fixed cost for each method on a 15-node network.

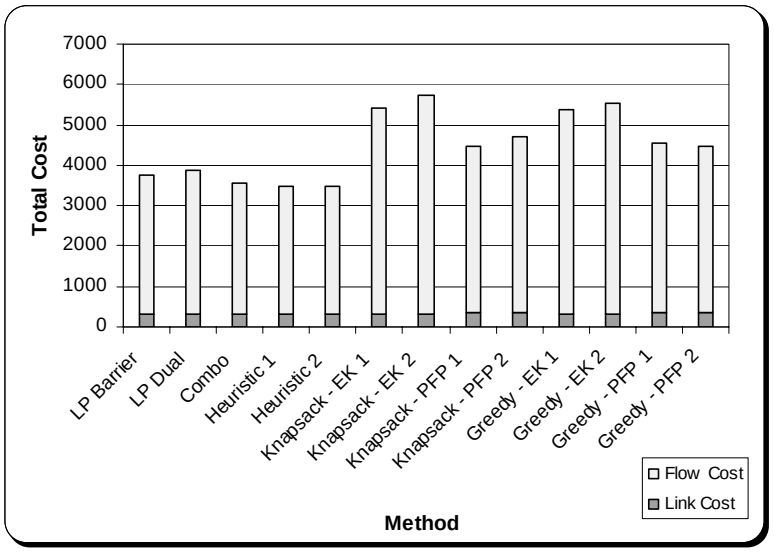

Figure 39. Total cost broken out by link and fixed cost for each method on a 20-node network.

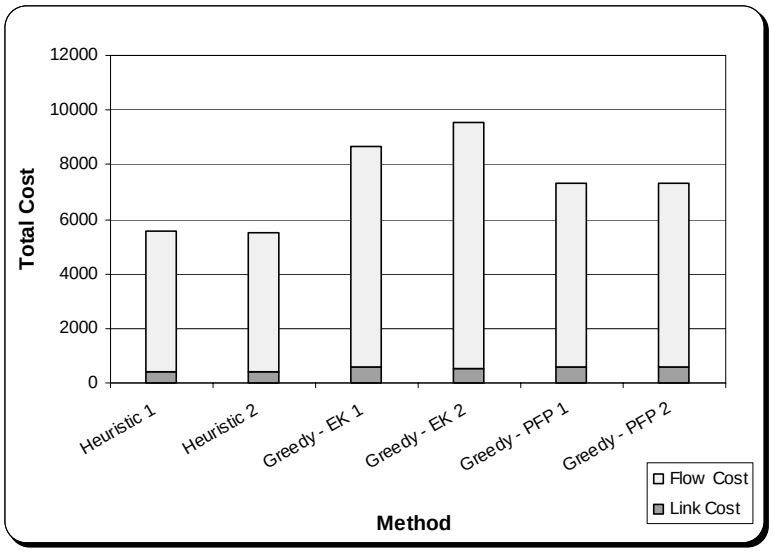

Figure 40. Total cost broken out by link and fixed cost for each method on a 25-node network.

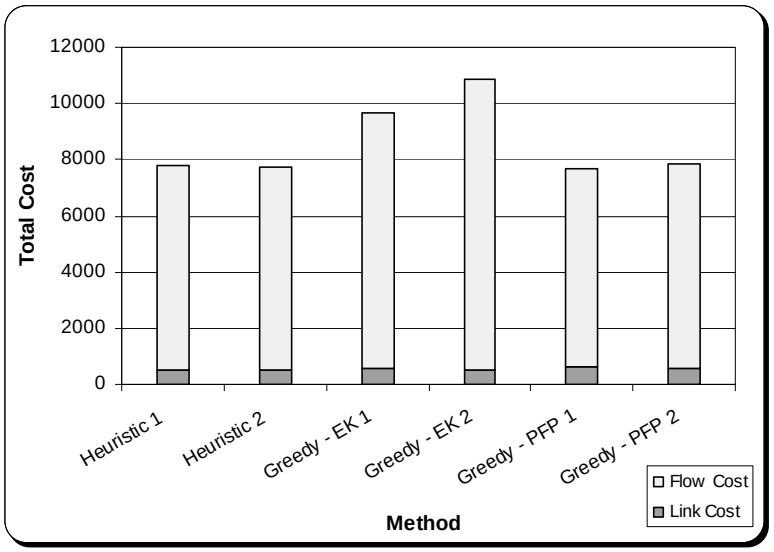

Figure 41. Total cost broken out by link and fixed cost for each method on a 30-node network.

Regardless of the method used or the size of the network, the link cost is generally comparable across the board. What separates the optimal solution from others however is the flow cost. Both the greedy and the knapsack method's lackluster performance clearly illustrate the difference between the more efficient LP methods. While the observed cost for the Pre-flow Push algorithms is tolerable (on average only 24% higher than optimal cost), the observed cost for the Edmonds-Karp algorithms is probably not (another 26% higher than Pre-flow Push's average cost). Another important factor is that as the size of the network increases—causing the number of commodities to increase—the flow cost is expected to rise. Hence, it is not surprising that the flow cost is the driving factor behind the increase in total cost over larger networks.

Figure 42, the overall total cost across the board, initially suggests that the costs remain relatively consistent for 20-node networks and smaller. However, the cost for larger networks is a bit more variable, perhaps due to the input file used.

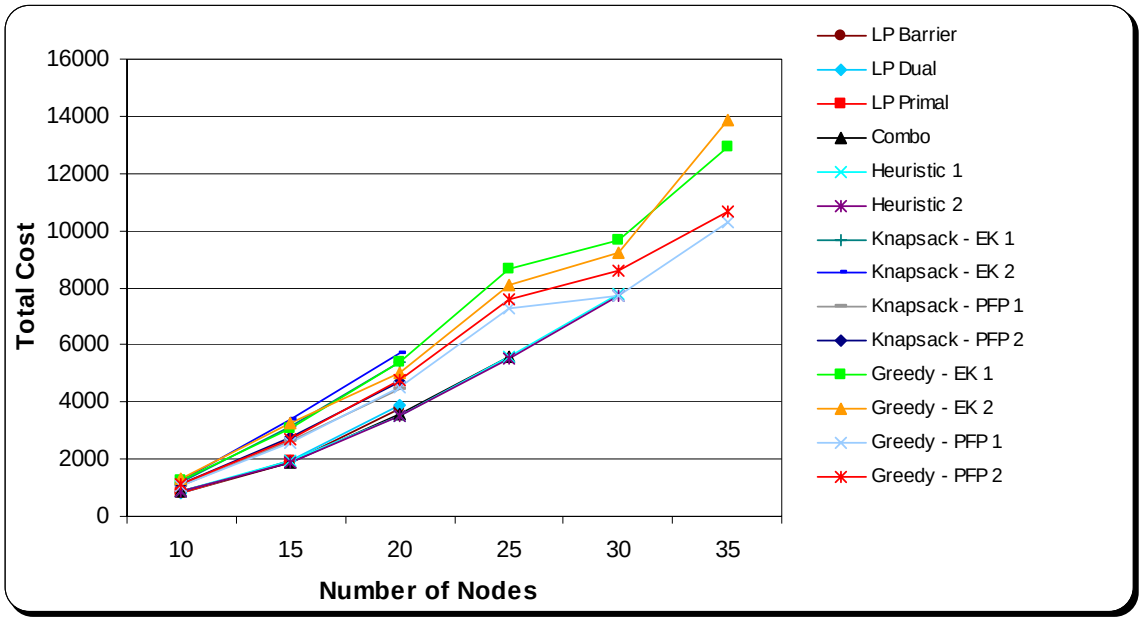

Figure 42. Overall total cost for all tested networks.

Metric 5: Network diameter

The final metric used to analyze the greedy and knapsack heuristics is network diameter. Network diameter is related to the number of hops per commodity, however, it provides a more general indication of how efficient a network is. As the number of nodes increases, the diameter is expected to remain relatively small as result of the randomness in picking edges. The results are depicted in Figure 43 – Figure 47.

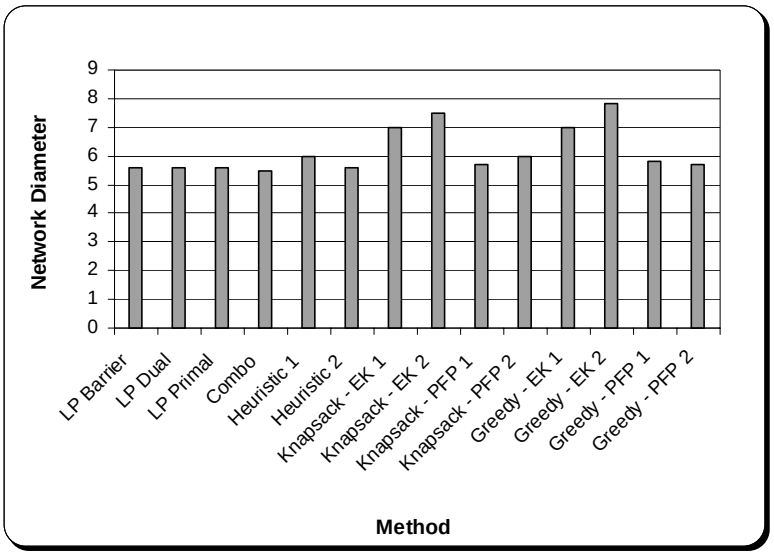

Figure 43. Network diameter for each method on a 10-node network.

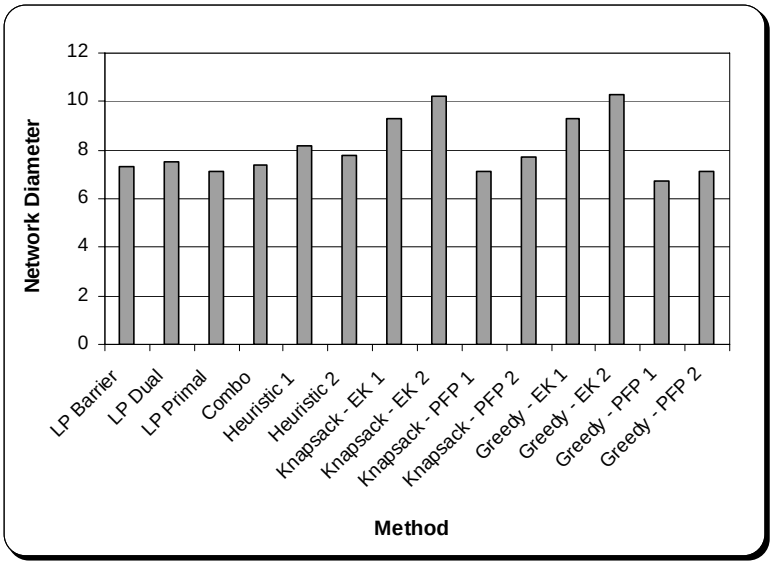

Figure 44. Network diameter for each method on a 15-node network.

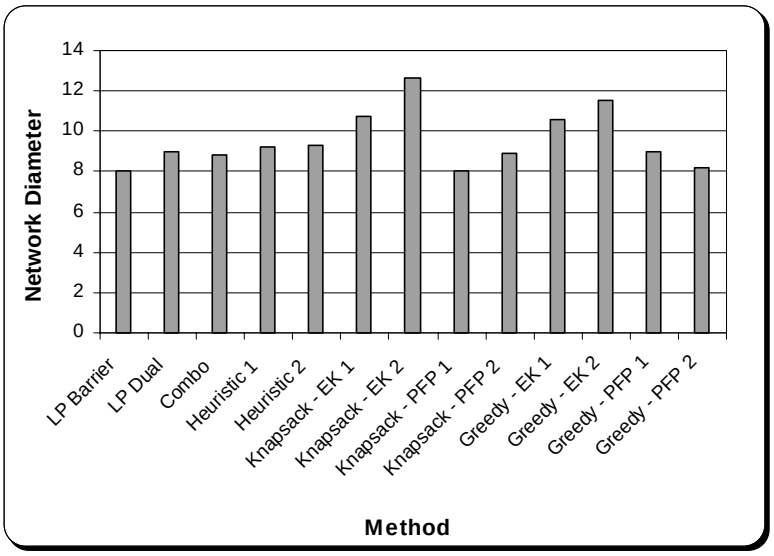

Figure 45. Network diameter for each method on a 20-node network.

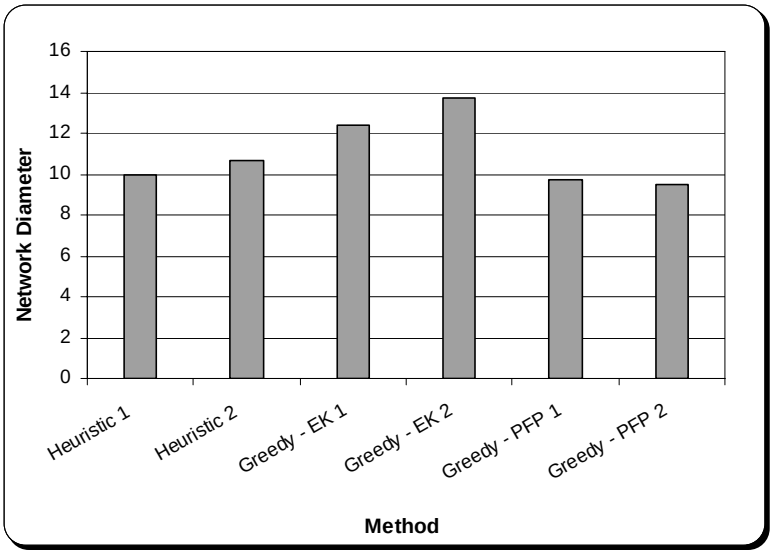

Figure 46. Network diameter for each method on a 25-node network.

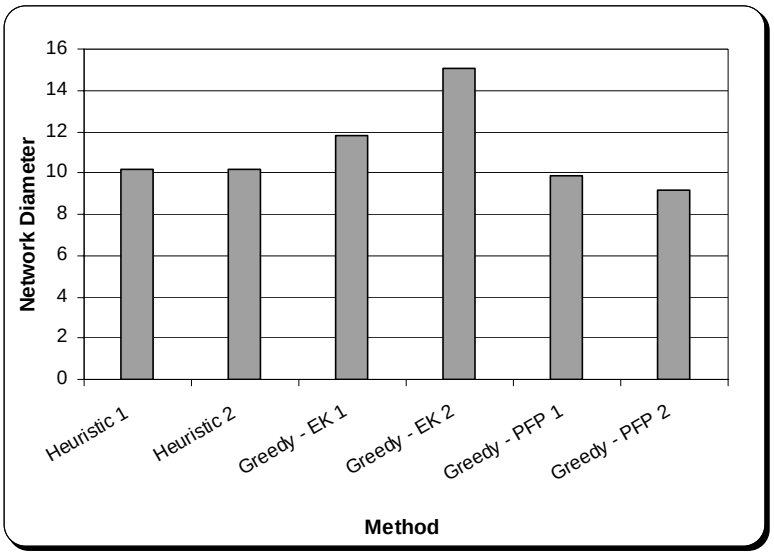

Figure 47. Network diameter for each method on a 30-node network.

A consistent theme among the results thus far has been that the Pre-flow Push algorithms generally outperform the Edmonds-Karp algorithms. The same holds for the network diameter. Additionally, there is only a small disparity between the methods suggesting the input file parameters (number of edges, nodes, etc) may dictate how big the network diameter more so than the algorithm itself. In some cases, the greedy Pre-flow Push heuristics outperform all of the others methods (15, 25, and 30 nodes). In most other cases however, the LP methods portray the best results—tied for best among the 20-node network was the LP Barrier method and the knapsack Pre-flow Push BFS heuristic.

Figure 48 illustrates the diameter of the network across the board. It is interesting to note the decrease in diameter from the 25-node network to the 30-node network. While there are only a few methods that are tested with this network, nearly all of them

exhibit the same behavior. As mentioned previously, this problem is a result of the network parameters.

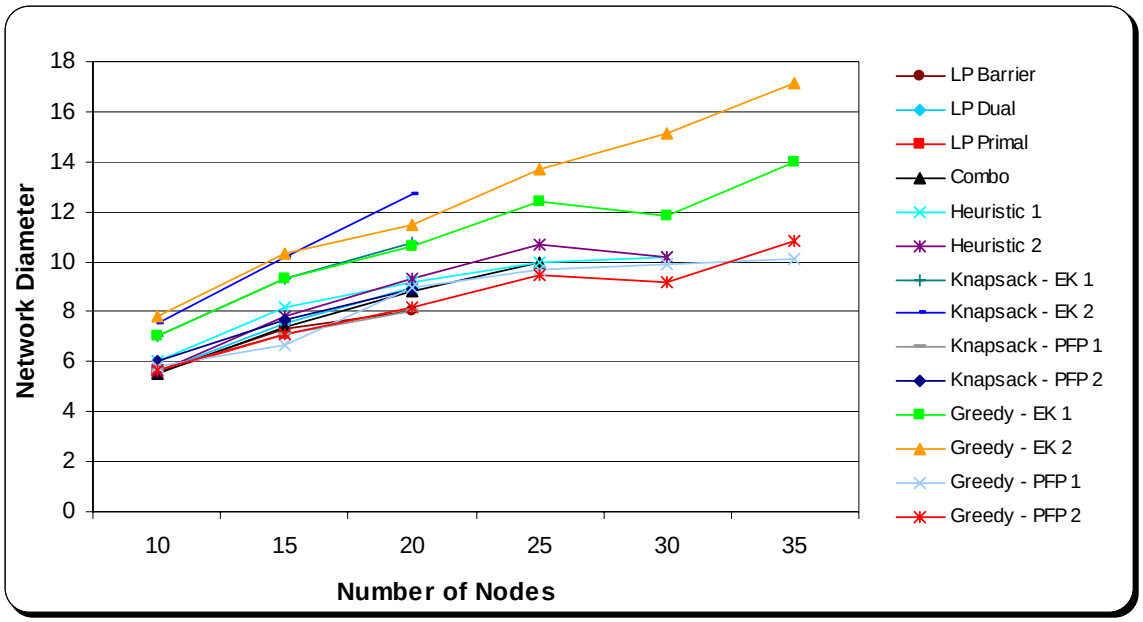

Figure 48. Overall network diameter for all tested networks.

Summary

This chapter provided an in-depth analysis of the experiments designed to compare and contrast the greedy and knapsack methods detailed in Chapter III with the MILP methods and heuristics used by Erwin. Furthermore, the results of each method were illustrated with various charts for the five key metrics: run time, number of hops, dropped commodities, total cost, and network diameter. While some results may yield inconclusive results, the general outcome suggests that the greedy and knapsack methods can be comparable, but often show evidence of less desirable characteristics. The final

chapter (V) provides some concluding remarks along with recommendations for future research.

V. Conclusions and Recommendations

Chapter Overview

This chapter provides a clear and concise summary of the research undertaken and described throughout the preceding chapters. Additionally, conclusions are revisited to emphasize the implications with which they present for related research. Lastly, a few recommendations for prospective research are offered.

Brief Review

Chapter I (Introduction) provided a general introduction to the topology control problem. The topology control problem is an NP-Hard problem in which a feasible topology is calculated while adhering to a number of constraints and limitations. Furthermore, Chapter I outlined the importance of the research and established goals with which motivated the undertaking even more so. Tomorrow's military networks will be highly mobile and predominantly ad hoc. Technology is already advancing in the areas of routing, fault tolerance, and connectivity. Thus, research is needed to develop flexible networks that are capable of sustaining a high operations tempo with minimal degradation in service and to explore these new concepts.

Chapter II (Literature Review) introduced the necessary background information required to comprehend the problem and the solution researched. In this case, flow networks played a significant role in the solution process. Maximum flow algorithms were also explored providing two candidates for the heuristic searches, Edmonds-Karp and Pre-flow Push, which set the stage for Chapter III. The last section provided a

review of the significant headway achieved by others who have performed similar research.

Chapter III (Methodology) described the inner details of the process used to generate solutions to the topology control problem. In this chapter, a small, real-world example illustrated the particulars of the problem. Then, a thorough account of the process for choosing commodities combinations using a greedy approach and the knapsack was explained, followed by the algorithmic specifications of the maximum flow algorithms. Lastly, the idea of potential edges was introduced identifying the important role they played.

Chapter IV (Analysis and Results) outlined the experimental design used to generate meaningful data which could then be compared and contrasted against previous research. Once the design process was explained, the limiting factors for the research were briefly discussed followed by a comprehensive and comparative analysis of the results for the five metrics. Recall that the primary methods examined afforded little performance gain as compared with the Erwin's MILP techniques and his heuristics.

Conclusions of Research

This research began with the goal of obtaining a reasonable solution to the topology control problem with running times ideally no greater than the cube of the size of the network. With proven order analyses of $O_{EK}(V \times E^2)$ and $O_{PFP}(V^3)$, respectively, the Edmonds-Karp and Pre-flow Push algorithms appeared very suitable contenders for just this problem. Implementing these algorithms proved to be a bit of a challenge as care had to be taken to accommodate for multiple commodities and potential edges. A

significant limiting factor, however, was memory utilization. The dynamic knapsack formulation used in this research provided a solution in pseudo-polynomial time, but due to the nature of the data structures used, a significant amount of memory was required to store the network state. Thus, four of the eight methods explored (the knapsack heuristics) suffered severe set backs because adjustments had to be made to circumvent the memory problem. It is very apparent that the trade-off for pseudo-polynomial time is memory.

Luckily, the four greedy heuristics were not tied to the memory utilization problem because they did not store residual networks. The trade-off was solution quality, and while greedy methods generally provide a good approximation, they are also subject to local minima traps.

In general, the performance of the eight heuristics was at least comparable to Erwin's methods. In various test networks, desirable results were collected for some of the metrics. However, by consolidating the data into a single chart that depicted the results for each network, the prevalent theme was that the greedy and knapsack heuristics were slightly outperformed in most cases either by the MILP methods or the Erwin's heuristics. Thus, some recommendations are provided such that follow-on research can explore other avenues that might lead to more desirable results.

Recommendations for Future Research

One of the main problems observed throughout this research was defining the problem in a way that models the real world with meaningful data. For example, rather than assigning the number of commodities arbitrarily, consider taking statistics on real

world network to better understand the traffic demands that are present in communications networks. Similarly, steps should be taken to appropriately define the characteristics of traffic requirements.

Another idea for subsequent research is to build upon the greedy approach by trying to alleviate the problem of getting stuck in local optima. One possible solution is to identify some threshold value that tracks the amount of lost benefit observed caused by the addition of any particular commodity. If this issue can be circumvented, the greedy method would be a plausible solution.

Sometimes, however, sticking to a single method is not always the best case. As several different approaches are researched, advantages and disadvantages alike become evident for each. Perhaps by utilizing a combination of multiple techniques, a better solution could be found. One example is to use Erwin's MILP for initial construction of a template network. Then, the greedy heuristics described in this research to quickly approximate a solution from the template network. Then, a GA approach such as the MOEA strategy described by Kleeman, et al. could be used to further refine a population of similarly approximated solutions. In essence, it is comparable wrapping each method within the scope of the previous. Such an approach would have to consist of an interface between the different methods such that information can be collectively shared. Figure 49 illustrates the proposal.

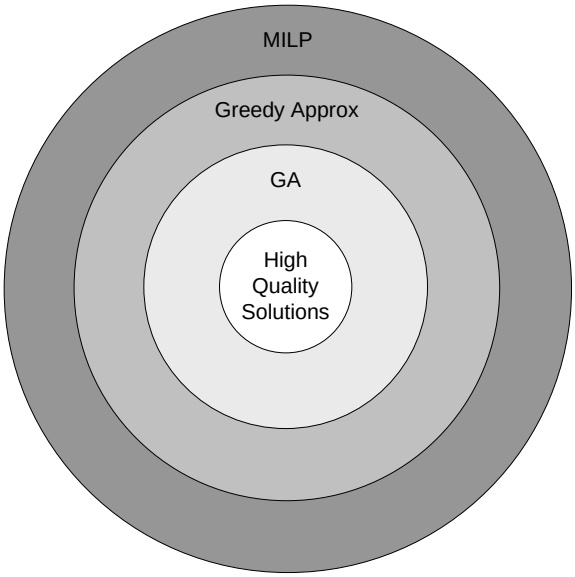

Figure 49. Combination of multiple methods used together.

While calculating solutions via simulations is useful, implementing these concepts in a real world testbed could provide enormous insight into the feasibility of the problem. A picture says a thousand words, and a real-life demonstration can go even further.

Lastly, this research, as well much of the previous studies, have centered around centralized algorithms. Such algorithms require complete (global) state information for each node in network. Often times, this information may not be available or may be costly to retrieve on a recurring basis. Therefore, it is recommended that a distributed implementation be explored. Decentralized, or distributed, systems have the freedom of existing independently of a central authority. Decisions are made with only the local data available to each constituent member of the system. Due to the mobility and robustness of future military networks, distributed systems concepts will be essential for operating with fast, flexible, and effective communications.

Summary

This chapter provided a brief summary of the work described in this document. After a review of the previous chapters, closing remarks were given drawing upon the conclusions found in Chapter IV. The salient features and limiting factors were revisited to emphasize the importance of those conclusions. Lastly, a variety of recommendations were imparted as a starting point for prospective research.

Appendix A

Table 5. Average performance statistics for each method on a 10-node network.

10 Nodes - Averages							
Method	**link cost**	**flow cost**	**total cost**	**number of hops**	**network diameter**	**dropped comm**	**run time (seconds)**
LP Barrier	161.30	662.94	824.24	2.07	5.60	0.00	20.89
LP Dual	161.30	662.94	824.24	2.07	5.60	0.00	17.97
LP Primal	161.30	662.94	824.24	2.07	5.60	0.00	19.31
Combo	139.60	721.42	861.02	2.12	5.50	1.30	1.72
Heuristic 1	144.50	742.83	887.33	2.20	6.00	1.20	0.48
Heuristic 2	143.90	725.10	869.00	2.22	5.60	3.00	0.54
Knapsack - EK 1	125.60	1067.10	1192.70	3.49	7.00	0.80	6.04
Knapsack - EK 2	133.90	1240.80	1374.70	3.85	7.50	0.30	7.21
Knapsack - PFP 1	168.40	918.30	1086.70	3.06	5.70	0.00	11.29
Knapsack - PFP 2	162.60	915.40	1078.00	3.08	6.00	0.20	12.69
Greedy - EK 1	127.10	1128.40	1255.50	3.56	7.00	0.50	1.76
Greedy - EK 2	144.80	1205.50	1350.30	3.86	7.80	0.80	2.24
Greedy - PFP 1	177.90	902.20	1080.10	3.06	5.80	0.10	3.48
Greedy - PFP 2	173.30	898.70	1072.00	3.05	5.70	0.30	4.80

Table 6. Average performance statistics for each method on a 15-node network.

15 Nodes - Averages							
Method	**link cost**	**flow cost**	**total cost**	**number of hops**	**network diameter**	**dropped comm**	**run time (seconds)**
LP Barrier	302.20	1579.01	1881.21	2.38	7.30	5.80	707.27
LP Dual	302.00	1616.31	1918.31	2.41	7.50	5.70	762.57
LP Primal	291.80	1638.60	1930.40	2.42	7.10	5.80	867.87
Combo	264.50	1602.55	1867.05	2.40	7.40	8.60	142.94
Heuristic 1	245.30	1703.92	1949.22	2.55	8.20	16.80	3.56
Heuristic 2	250.30	1640.68	1890.98	2.51	7.80	19.00	3.64
Knapsack - EK 1	259.90	2875.60	3135.50	4.02	9.30	1.90	63.41
Knapsack - EK 2	273.60	3312.20	3585.80	4.46	10.20	6.30	104.95
Knapsack - PFP 1	320.70	2289.80	2610.50	3.35	7.10	0.90	152.39
Knapsack - PFP 2	310.30	2274.70	2585.00	3.30	7.70	1.10	186.72
Greedy - EK 1	262.10	2819.30	3081.40	3.97	9.30	2.90	26.27
Greedy - EK 2	256.20	3275.20	3531.40	4.46	10.30	6.40	32.84
Greedy - PFP 1	323.50	2217.80	2541.30	3.29	6.70	1.40	50.02
Greedy - PFP 2	315.00	2234.40	2549.40	3.31	7.10	2.90	65.26

Table 7. Average performance statistics for each method on a 20-node network.

20 Nodes - Averages							
Method	**link cost**	**flow cost**	**total cost**	**number of hops**	**network diameter**	**dropped comm**	**run time (seconds)**
LP Barrier	303.00	3448.25	3751.25	2.66	8.00	29.00	2008.75
LP Dual	326.00	3551.18	3877.18	2.81	9.00	25.00	2321.96
LP Primal	no data provided						
Combo	328.10	3227.03	3555.13	2.63	8.80	17.40	641.20
Heuristic 1	314.00	3177.69	3491.69	2.68	9.20	38.50	24.40
Heuristic 2	315.70	3168.16	3483.86	2.71	9.30	40.30	17.56
Knapsack - EK 1	331.78	5068.78	5400.56	4.19	10.78	47.11	6315.95
Knapsack - EK 2	327.67	5407.11	5734.78	4.77	12.67	75.89	33360.56
Knapsack - PFP 1	361.22	4118.56	4479.78	3.56	8.00	48.89	25955.00
Knapsack - PFP 2	343.25	4374.75	4718.00	3.65	8.88	34.75	14160.11
Greedy - EK 1	328.20	5059.30	5387.50	4.27	10.60	52.40	142.47
Greedy - EK 2	324.50	5202.20	5526.70	4.68	11.50	80.80	183.15
Greedy - PFP 1	370.10	4163.00	4533.10	3.59	9.00	41.30	379.51
Greedy - PFP 2	356.00	4096.90	4452.90	3.61	8.20	57.20	502.54

Table 8. Average performance statistics for each method on a 25-node network.

25 Nodes - Averages							
Method	**link cost**	**flow cost**	**total cost**	**number of hops**	**network diameter**	**dropped comm**	**run time (seconds)**
LP Barrier LP Dual LP Primal Combo	no data provided						
Heuristic 1	425.10	5151.47	5576.57	2.78	10.00	79.50	135.65
Heuristic 2	431.20	5072.25	5503.45	2.77	10.70	78.80	291.33
Knapsack - EK 1 Knapsack - EK 2 Knapsack - PFP 1 Knapsack - PFP 2	not enough data collected due to undesirable performance						
Greedy - EK 1	558.80	8117.20	8676.00	4.56	12.40	107.30	633.42
Greedy - EK 2	532.80	9020.20	9553.00	5.25	13.70	143.90	833.28
Greedy - PFP 1	607.80	6680.40	7288.20	3.68	9.70	74.70	1682.91
Greedy - PFP 2	585.70	6704.60	7290.30	3.75	9.50	83.80	2020.81

Table 9. Average performance statistics for each method on a 30-node network.

30 Nodes - Averages							
Method	**link cost**	**flow cost**	**total cost**	**number of hops**	**network diameter**	**dropped comm**	**run time (seconds)**
LP Barrier	no data provided						
LP Dual							
LP Primal							
Combo							
Heuristic 1	491.40	7277.95	7769.35	2.80	10.20	197.50	1101.31
Heuristic 2	503.80	7241.95	7745.75	2.84	10.20	181.30	864.49
Knapsack - EK 1	not enough data collected due to undesirable performance						
Knapsack - EK 2							
Knapsack - PFP 1							
Knapsack - PFP 2							
Greedy - EK 1	564.70	9118.90	9683.60	4.54	11.80	316.20	1833.91
Greedy - EK 2	528.20	10327.60	10855.80	5.41	15.10	371.00	2441.80
Greedy - PFP 1	598.60	7097.00	7695.60	3.80	9.90	338.60	8558.60
Greedy - PFP 2	588.10	7282.30	7870.40	3.80	9.20	327.70	9202.70

Table 10. Average performance statistics for each method on a 35-node network.

35 Nodes - Averages							
Method	**link cost**	**flow cost**	**total cost**	**number of hops**	**network diameter**	**dropped comm**	**run time (seconds)**
LP Barrier	no data provided						
LP Dual							
LP Primal							
Combo							
Heuristic 1							
Heuristic 2							
Knapsack - EK 1	not enough data collected due to undesirable performance						
Knapsack - EK 2							
Knapsack - PFP 1							
Knapsack - PFP 2							
Greedy - EK 1	758.13	12157.63	12915.75	4.78	14.00	500.00	5519.13
Greedy - EK 2	689.14	13178.43	13867.57	5.66	17.14	587.14	7751.43
Greedy - PFP 1	783.29	9521.86	10305.14	3.79	10.14	475.71	28144.43
Greedy - PFP 2	774.14	9906.14	10680.29	3.93	10.86	473.29	29932.29

Bibliography

1. Gettle, M., MSgt (USAF). *Air Force releases new mission statement*. Air Force Print News December 8, 2005 January 6, 2007 [cited January 6, 2007]; Available from: http://www.af.mil/news/story.asp?id=123013440.

2. Wynne, M.W. and T.M. Mosely, Gen. (USAF). *SECAF/CSAF Letter to Airmen: Mission Statement*. December 7, 2005 [cited January 6, 2007]; Available from: http://www.af.mil/library/viewpoints/secaf.asp?id=192.

3. Lopez, T.C., SSgt (USAF) *Air Force leaders to discuss new 'Cyber Command'*. Air Force Print News December 5, 2006 [cited 6 January 2007]; Available from: http://www.af.mil/news/story.asp?storyID=123028524.

4. Director for Strategic Plans and Policy, *Joint Vision 2020. America's Military: Preparing for Tomorrow*, DoD, Editor. June 2000, US Government Printing Office.

5. Hopkinson, K.M. and S.R. Graham, Maj (USAF) *Annual NRO Presentation Slides*, in *Microsoft PowerPoint*. 2006, Air Force Institution of Technology: WPAFB, OH.

6. Davis, C.C., I.I. Smolyaninov, and S.D. Milner, *Flexible optical wireless links and networks.* Communications Magazine, IEEE, 2003. **41**(3): p. 51-57.

7. Arpacioglu, O. and Z.J. Haas. *On the scalability and capacity of wireless networks with omnidirectional antennas*. 2004.

8. Hochbaum, D.S., ed. *Approximation Algorithms for NP-Hard Problems*. 1997, PWS Publishing Company: Boston, MA.

9. Kleeman, M.P., et al., *Solving Multicommodity Capacitated Network Design Problems using a Multiobjective Evolutionary Algorithm*. 2007, Air Force Institution of Technology.

10. Russell, S.J. and P. Norvig, *Artificial Intelligence: A Modern Approach*. 2 ed. 2003, Upper Saddle River, NJ: Prentice-Hall.

11. Hartlage, R.B., *An Efficient Metaheuristic for Dynamic Network Design and Message Routing*, in *Operations Research*. 2007, Air Force Institute of Technology: WPAFB, OH.

12. Rajaraman, R., *Topology Control and Routing in Ad hoc Networks*, College or Computer Science: Boston, Ma. p. 14.

13. Erwin, M.C., *Combining Quality of Service and Topology Control in Directional Hybrid Wireless Networks*, in *Operations Research*. 2006, Air Force Institute of Technology: WPAFB, OH. p. 117.

14. Busse, M., et al. *TECA: A topology and energy control algorithm for wireless sensor networks*. in *Proceedings of the 9th ACM international Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems*. 2006. Terromolinos, Spain: ACM Press, New York, NY.

15. Santi, P., *Topology control in wireless ad hoc and sensor networks.* ACM Computing Surveys, 2005. **37**(2): p. 164 - 194.

16. Corner, J.J. and G.B. Lamont. *Parallel simulation of UAV swarm scenarios*. in *Proceedings of the winter Simulation Conference*. 2004.

17. Llorca, J., et al. *Optimizing Performance of Hybrid FSO/RF Networks in Realistic Dynamic Scenarios*. in *Proceedings of the SPIE Optics and Photonics*. July 2005.

18. Kleinberg, J. and É. Tardos, *Algorithm Design*. 2006, Addison-Wesley: Boston, MA. p. 337 - 451.

19. Schrijver, A., 2002, *On the history of the transportation and maximum flow problems.* Math Programming, 2002. **91**(3).

20. Cormen, T.H., et al., *Introduction to Algorithms*. 2001, MIT Press, McGraw-Hill: Cambridge and Boston, MA. p. 643 - 701.

21. Ford, L.R., Jr and D.R. Fulkerson, *Flows in networks*. 1962, Princeton, NJ: Princeton University Press.

22. Edmonds, J. and R.M. Karp, *Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems.* ACM Computing Surveys, 1972. **19**(2): p. 248 - 264.

23. Lewis, H.R. and L. Deneberg, *Data Structures and Their Algorithms*. 1991, Addison-Wesley: xxx. p. 452 - 462.

24. Ahuja, R.K., T.L. Magnanti, and J.B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. 1993, Upper Saddle River, NJ: Prentice-Hall.

25. Associates, D., *Xpress-Optimizer Optimization Software*. 2001.

26. Bertsimas, D. and J.N. Tsitsiklis, *Introduction to Linear Optimization*. 1997, Belmont, MA: Athena Scientific.

27. Desai, A. and S. Milner, *Autonomous Reconfiguration in Free-Space Optical Sensor Networks.* Selected Areas in Communications, IEEE Journal on, 2005. **23**(8): p. 1556-1563.

28. Wikipedia Contributors. *Complete Graph*. 2006 [cited June 4, 2006]; Available from: http://en.wikipedia.org/wiki/Complete_graph.

29. Cherkassky, B.V. and A.V. Goldberg, *On Implementing Push-Relabel Method for the Maximum Flow Problem.* Algorithmica, 1997. **19**: p. 390 - 410.

30. Weisstein, E.W. *Power Set*. 2006 [cited October 12, 2006]; MathWorld--A Wolfram Web Resource]. Available from: http://mathworld.wolfram.com/PowerSet.html.

31. Garey, M.R. and D.S. Johnson, *Computers and Intractability: A guide to the Theory of NP-Completeness*. 1979, W. H. Freeman and Company: New York, NY. p. 65.

32. Wikipedia Contributors. *Knapsack Problem*. 2007 [cited January 30, 2007]; Available from: http://en.wikipedia.org/w/index.php?title=Knapsack_problem

33. Levitin, A.V., *Introduction to the Design and Analysis of Algorithms*. 2002, Addison-Wesley Longman Publishing Co., Inc.: Boston, MA. p. 303 - 330.

Vita

Captain Roger Lance Garner graduated from Bryan Station High School in Lexington, Kentucky in May 1998. He entered undergraduate studies at Transylvania University where he graduated with Bachelor of Arts degree in Computer Science in May 2002. He was commissioned through Detachment 290 AFROTC at the University of Kentucky.

His first assignment to the 721st Air Mobility Operations Group (AMOG) where he served as Deputy Chief for European En Route Communications and Executive Officer to the Commander. In June 2005, he entered the Computer Science degree program at the Graduate School of Engineering And Management, Air Force Institute of Technology. Additionally, Capt Garner was inducted in to the national engineering honor society, Tau Beta Pi. Upon graduation in March 2007, he will be assigned to the College of Aerospace Doctrine Research and Education at Maxwell Air Force Base, Montgomery, Alabama where he will be the Chief of the Long-Term Integration Section for the Air Force Wargaming Institute.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
22032007	Master's Thesis	June 2005 – March 2007

4. TITLE AND SUBTITLE

HEURISTICALLY DRIVEN SEARCH METHODS FOR TOPOLOGY CONTROL IN DIRECTIONAL WIRELESS HYBRID NETWORKS

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Garner, Roger Lance, Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/07-03

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Office of Scientific Research (AFOSR)
Dr. David Luginbuhl
875 N. Randolph Street Arlington, VA 22203-1768, (703) 696-6207

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Information and Networked Communications play a vital role in the everyday operations of the United States Armed Forces. This research establishes a comparative analysis of the unique network characteristics and requirements introduced by the Topology Control Problem (also known as the Network Design Problem). Previous research has focused on the development of Mixed-Integer Linear Program (MILP) formulations, simple heuristics, and Genetic Algorithm (GA) strategies for solving this problem. Principal concerns with these techniques include runtime and solution quality. To reduce runtime, new strategies have been developed based on the concept of flow networks using the novel combination of three well-known algorithms; knapsack, greedy commodity filtering, and maximum flow. The performance of this approach and variants are compared with previous research using several network metrics including computation time, cost, network diameter, dropped commodities, and average number of hops per commodity. The results conclude that maximum flow algorithms alone are not quite as effective as previous findings, but are at least comparable and show potential for larger networks.

15. SUBJECT TERMS

Network Topology, Communications Network, Directional Antennas, Network Flows, Topology Control, Commodity, Hybrid Transceivers, Wireless, Edmonds-Karp, Pre-flow Push, Maximum Flow, Greedy, Knapsack

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr Kenneth M. Hopkinson
U	U	U	UU	101	19b. TELEPHONE NUMBER *(Include area code)* (937) 785-3636, ext 4579 (Kenneth.hopkinson@afit.edu)