Allocating Virtual and Physical Flows for Multiagent Teams in Mutable, Networked Environments

Steven Okamoto

CMU-CS-12-129

August 2012

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Katia Sycara, Chair
Srinivasan Seshan
Paul Scerri
Milind Tambe, University of Southern California

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was funded in part by the AFOSR MURI grant FA9550-08-1-0356, in part by ONR MURI N000140811186, and in part by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-06-3-0001.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government.

Keywords: multiple source network flow, partial centralization, network augmentation, network security game

Abstract

The movement of information, agents, and resources is a crucial part of cooperative multiagent systems: decision makers must receive data in a timely manner to make good decisions, while agents and resources must be provided at appropriate locations for tasks to be completed. Flow allocation meets these conditions by computing paths through the environment, be it the communication network (for data or software agents) or the physical world (for embodied agents or physical resources). This thesis addresses the problem of allocating flows when the environment is mutable, either by the agents or by a malicious adversary.

In this thesis I represent the environment as a graph with agents and tasks represented by source and sink nodes, respectively. The agents are partitioned into groups, and the flows from members of a group must be sent to the same sink, reflecting cases where a task requires multiple agents to work together or a decision requires multiple inputs. Capacity and cost constraints on the graph reflect environmental factors affecting the feasibility and quality of allocations. I prove that even relatively simple problems of allocating flows for groups in graphs are not just hard to solve optimally, but also hard to approximate.

In some cases the agent team can add additional nodes and edges to the graph, for instance by deploying unmanned aerial vehicles (UAVs) as communication relays to supplement a ground-based wireless ad hoc network. I prove that augmenting the network optimally is intractable, and provide algorithms for solving it heuristically.

In other cases an adversary can impair the agents by choosing attacks that increase costs to the agents. Agents must then choose their paths through the network strategically, reasoning not just about the environmental costs but also about the behavior of the adversary. I formalize this interaction as a game between the agent team and the adversary, and provide polynomial time algorithms for computing equilibrium strategies in simultaneous zero-sum games, simultaneous non-zero-sum games where each player unilaterally incurs costs to play their strategies, and sequential non-zero-sum games where the sender can commit to a strategy.

Acknowledgments

My path to completing my thesis has been a long and twisting one but I've been fortunate to have had many mentors, colleagues, and friends accompany me along the way. I am grateful to my advisor, Katia Sycara, who has been with me the whole way offering guidance and unflagging support; to Paul Scerri and Milind Tambe for their keen insights and advice that they have offered me through the years, even before serving on my committee; and to Srinivasan Seshan who brought a fresh and valuable perspective as a member of my committee outside of my usual area of expertise.

I want to acknowledge all of the talented researchers I've had the opportunity to work with while here at CMU, including Noam Hazon, Praveen Paruchuri, Roie Zivan, Nathan Brooks, Robin Glinton, Sean Owens, Nilanjan Chakraborty, and Joseph Giampapa. Some of these collaborations bore academic fruit, others yielded only intellectual stimulation, and others were just a pleasant way to pass a spare afternoon. I am grateful for all of them.

My heartfelt thanks also goes out to those who have kept things running smoothly in the background, especially Deb Cavlovich and Marliese Bonk.

I would never have made it without the love and support of my family and friends. My parents, Alan and Charlene Okamoto, and sister, Amy Maneki, never stopped believing in me ... or asking when I would next visit. Their constant encouragement, inspiration, and humor kept me going. My friends — Taiki Esheim, Natasha Capellas, Jennifer Salcedo, Beccy Aldrich, Anna Grauerholz, Amy Fernandez, Matt Hessing, and many others — kept me sane through the years. Most of all, I want to thank Amanda Mitchell, whose love is still the greatest thing I've earned.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A key feature of cooperative multiagent systems is the movement of agents, information, and resources in order to facilitate task execution. Performing each task requires a combination of agents, information, and resources, and these elements must generally be brought to a single location for the task to be performed successfully. For example, in a disaster response application, extinguishing a fire (the task) requires bringing fire trucks (the agents) and water (the resource) to the location of the fire. In a sensor network application, sensor readings (information) from multiple sensors (the agents) must be sent to a base station (the resource) for data fusion to track a target (the task). Different paths can impose different costs to task execution, for example by draining more battery power in the sensor network or delaying arrival of the fire trucks to the fire. The problem of choosing paths for the movement of agents, information, and resources is thus crucial because it governs both what tasks can be performed as well as the quality with which they are performed.

Network flows provide a natural way to represent this problem. The environment is represented as a finite, directed graph with nodes representing locations and edges representing allowable transitions between locations. This graph may model the external, physical environment, such as the buildings and road network of a city, or the rooms and corridors of a building, or it may model the communication network between agents or some other internal structure of the team. Source nodes represent the initial locations of things that must move through the graph, such as agents or information, while sink nodes represent the locations of task execution. Capacity and cost constraints on edges reflect environmental factors that affect feasibility and quality of allocations, with capacitated graphs best suited for modeling communication networks and costly graphs appropriate for both physical and communication environments. For example, in a physical environment costs may model distances traveled by robots, while in a communication environment costs reflect

the battery usage for transmitting needed data over communication links, while capacity constraints represent limited available bandwidth on those links. The problem of meeting the requirements for task execution then becomes the problem of allocating flows from the source nodes to the sink nodes.

This thesis addresses the problem of flow allocation in multiagent teams when the environment is mutable, either by the agents or by a malicious adversary. The type of modification depends on whether the environment is capacitated or costly. In capacitated environments agents may relieve bottlenecks or provide connectivity to disconnected parts of the environment by augmenting the environment with additional nodes and edges. For example, in a communication network this may be achieved by deploying unmanned aerial vehicles to act as communication relays to supplement a ground-based ad hoc wireless network. In costly environments, an adversary can impose additional costs, for example by deploying cameras to observe the movements of agents, placing obstacles to delay agents, or jamming a communication network to force retransmissions.

1.1 Flow Allocation for Multiagent Teams

A common approach to designing multiagent teams is to separate domain-level task knowledge from domain-independent coordination ability. Domain-level knowledge is typically represented using hierarchical task networks (HTNs) [18] that explicitly specify dependencies betweens tasks, including the requirements for execution. Tasks in an HTN may be primitive tasks that can be directly executed, goal tasks that describe desired states of the world, or compound tasks that are made up of other tasks. In multiagent systems, multiple agents must often work together to achieve goal tasks and compound tasks, introducing additional constraints between tasks not found in task networks designed for single agent planning. This shortcoming has been addressed by task networks specifically tailored to multiagent teams, such as TAEMS [16], SharedPlans [24], and team-oriented programs [53].

Domain-independent multiagent infrastructures such as GPGP [37], RETSINA [60], or Machinetta [58] provide reusable software components implementing algorithms to address problems such as planning and scheduling, subteam formation, and task and resource allocation. Planning identifies the domain-level tasks the team will execute in order to achieve high level goals, based on the requirements for task execution and their effects represented in the task networks. Most tasks require only a small fraction the agents and resources of the full team, and so the agents are organized into subteams that coordinate closely. Task allocation is the problem of choosing the specific tasks for agents to execute.

Flow allocation has a great deal of overlap with these problems. The plan determines the sink nodes on which flow allocation depends. Conversely, knowing how or even if task requirements can be met, for example if data from sensor nodes can be transmitted to a base station for fusion, is crucial for planning. Subteam formation affects the choice of source nodes while task allocation affects which sink nodes the flows should go; it is clear that the choice and quality of flows through the environment will depend on the both of these factors. However, this overlap with other multiagent problems is not unique to flow allocation; instead, the extent to which flow allocation interacts with the other problems just reflects the degree to which those problems interact with each other.

While there is no universally accepted way to resolve these interdependencies, in this thesis we assume that flow allocation occurs after planning and subteam formation. As a consequence, the set of source nodes and sink nodes are known, and the source nodes are partitioned into disjoint *groups* corresponding to the subteams partitioning the agents. All source nodes in a group must send flow to the same sink. This corresponds to the subteam being assigned a single task, which may be a compound task for a subteam with multiple agents. For example, in a disaster response domain [35], a subteam may be composed of several fire trucks and ambulances assigned the task of responding to a burning building. This compound task in turn has component tasks assigned to each individual fire truck or ambulance to put out the fire and to rescue survivors. From the flow allocation perspective it is sufficient to consider the compound task that involves a single location for task execution of all members of the subteam.

We do allow for overlap with a limited form of task allocation. In many of the problems we consider the flow allocation problem includes choosing a sink for each group as well as choosing the flow from the members of the group to that sink. This corresponds to choosing a task execution location for each group. With physical tasks, sinks represents the location of the task in the physical environment, like the burning building in the stricken city; agents must physically move through the environment to reach the task location. Thus choosing a sink represents assigning the task of responding to a particular burning building to the subteam. With computational tasks, the source nodes represent the locations of input data, while the sinks represent locations where the computation can be performed, and the flows correspond to streams of data through the communication network. For example, any base station could perform the data fusion task for a group of sensor nodes provided it receives all of the sensor readings. Selecting a sink corresponds to selecting a base station to perform the data fusion.

In addition to facilitating execution of domain-level tasks, flow allocation may also be applicable to control-level computational tasks that arise from the team's coordination algorithms. A number of multiagent coordination algorithms require groups of agents to

transmit data to a single location for computation. Examples of such control-level computational tasks include winner selection for auction-based algorithms [22], partial centralization in distributed constraint optimization [41, 51], and automated monitoring of team plan execution [57]. Flow allocation can be used to optimize the choice of auctioneer, centralization point, or plan monitor.

1.2 Flow Allocation in Capacitated Environments

We first address the problem of flow allocation for task execution in capacitated environments. These are most natural for computational tasks in communication networks where the amount of data that can be transmitted is limited, but can also model physical environments with transportation limitations.

The environment is modeled as a directed graph with edge capacities, sink nodes represent task execution locations, and groups of source nodes represent subteams of agents. The source nodes of each group have an amount of flow that they must transmit to the sinks, and the flow for all members of the same group must go to the same sink. This corresponds to all of the required data for a computational task being sent to the same node for processing. The total amount of flow on each edge is subject to the capacity limit for that edge, and it may not possible for all groups to transmit their full amounts of flow to sinks. The flow allocation problem we address is to maximize the number of groups able to transmit their full amounts of flow. This corresponds to enabling the maximum number of subteams to execute their assigned tasks, a problem we formalize as the Maximum Satisfied Groups (MaxSG) problem.

Many well-known flow problems such as maximum flow or minimum cost flow can be solved efficiently by exploiting the divisibility of flows among multiple paths to meet capacity constraints. However, we show that MaxSG is intractable even when flows are divisible. The reason for this are two integral constraints. First, the source nodes in a group must all send their flow to the same sink. Choosing a sink for each group is part of the MaxSG problem, and it corresponds to selecting a single processing node on which to perform a computational task for a subteam of agents. Second, all of the flow from the source nodes of a group must be available at the selected sink in order for task execution to occur. We provide an algorithm for solving MaxSG optimally by expressing these constraints in a mixed integer linear program (MILP).

We also consider two variations the basic flow allocation problem. In the first, we restrict the graph of the environment to be a directed, rooted tree with a single sink at the root. This kind of functional topology arises from several proposed algorithms designed to simplify

or improve the efficiency of ad hoc wireless networks. We provide a pseudo-polynomial time algorithm (the running time depends polynomially on the magnitude of the largest capacity) for optimally solving MaxSG in such an environment.

In the second variation we require that the flows be unsplittable; that is, flow from a source node cannot be divided on multiple paths to the sink. This is useful for modeling physical agents like robots that may have a fixed size that restricts their ability to move through the environment, and cannot divide themselves into multiple smaller parts when confronted with a bottleneck. We show that MaxSG for unsplittable flows is also intractable, and moreover we prove that there is not a non-trivial guaranteed approximation algorithm unless P=NP.

1.3 Network Augmentation

The agents' ability to modify the environment is motivated by approaches to augment communication networks by the addition of supplemental nodes to improve connectivity and capacity [68, 38, 61]. Establishing and maintaining effective wireless communication networks is a major challenge for many multiagent domains, such as future military operations and disaster response teams. In these applications, agents often do not have access to established communication infrastructure, and so must rely on an ad hoc network to meet their communication needs. Limited communication ranges, communication capacity constraints, signal obstruction (by terrain, foliage, and buildings), agent mobility, and wireless interference restrict the effectiveness of ad hoc networks in such settings. Augmenting the network through the addition of additional nodes is one way to address these issues.

These supplemental nodes will often be large enough and have sufficient power resources that they can be equipped with sophisticated communication hardware with longer range and higher capacity than those carried by people on the ground or in low-power robots and sensors. For example, unmanned aerial vehicles (UAVs) and unmanned ground vehicles (UGVs) can be used for outdoor environments, small robots can be used for indoor settings, and additional base stations can be deployed in sensor networks, all with dedicated hardware with superior capabilities to the other nodes in the network. UAVs in particular have received attention because they possess additional advantages such as operating in largely obstruction-free airspace, which increases range, capacity, and reliability of communication among UAVs, often having clearer lines of sight between UAVs and agents on the ground, and being able to travel more quickly and freely than ground- based agents, which allow them to respond to agents' movements.

The central problem of network augmentation is determining where the supplemental nodes should be positioned. The supplemental nodes can be used in a variety of ways, for instance by providing connectivity between two disconnected parts of the network, improving capacity in parts of the network with heavy traffic, and even acting as processing nodes for computational tasks, due to their superior capabilities. The optimal positioning of supplemental nodes will therefore depend on the communication needs of the network. At the same time, the communication needs will be affected by the positioning, as new flows may become possible and sink locations change. Network augmentation thus adds an additional and interdependent level of optimization to the flow allocation problem.

Network augmentation therefore involves partial knowledge of both the topology and the communication needs. In considering flow allocation, we assumed a known topology but partial knowledge of the communication needs: we knew the source groups but not the sink assigned to each source group. Other work in network design typically assumes full knowledge of the communication needs in the form of flow demands for specific source-sink pairs, but only partial knowledge of the topology. In addressing network augmentation for flow allocation we must simultaneously address both forms of partial knowledge to find optimal solutions.

We represent the *initial network* without supplemental nodes as a graph with capacity constraints on edges. This can represent the effects of obstructions, interference, signal attenuation, and other factors in the properties of the network. In addition, we restrict the deployment of supplemental nodes to a finite set of possible locations whose connectivity properties are known, which is represented by the *potential network*. These locations may be pre-selected as desirable or acceptable, or generated as a finite approximation. In practice, these graphs can be generated in real-time if the relevant properties of the wireless network are known.

Flow allocation for source groups is performed on the *actual network* determined by a deployment of supplemental nodes to possible locations in the potential network, where possible locations (and incident edges) in the potential network that have not been filled by a supplemental node are removed. The supplemental nodes are able to act as sinks as well as relays, and we consider two related problems, the first of maximizing the number of satisfied groups given a fixed number of supplemental nodes and the second of finding the minimum number of supplemental nodes required for all groups to be able to be satisfied. We prove that these problems are intractable and provide optimal and heuristic algorithms for solving them.

1.4 Flow Allocation in Adversarial Environments

Some of the most common multiagent domains, those in military and law enforcement settings, are inherently hostile. The agents must operate in an environment that is not just the result of neutral processes but is influenced by a purposive adversary whose interests are opposed to those of the agents. This creates a very different setting for flow allocation than a neutral environment or one that can be affected through network augmentation by the agents. In those contexts, the problem is one of straightforward if computationally difficult optimization. In adversarial settings, flows must be chosen strategically even as the adversary is actively and rationally attempting to counter their efforts.

We consider costly environments where there are no capacity limitations, but transmitting flow on each edge incurs a cost for the agents that is proportional to the amount of flow sent. This can be represented by defining a cost vector over the edges of the graph and computing the total costs as the scalar product of the cost vector and the flow vector describing the amount of flow sent on each edge. This setting is widely applicable to both physical environments and communication environments. For example, costs may reflect the time it takes for a vehicle to move through a city, or battery depletion used to transmit across a communication link. The agent team seeks to minimize the total costs required to execute their tasks.

In the absence of an adversary, it is straightforward to compute satisfying flows for the source groups that minimize the total costs incurred by the agents, and this can be done in polynomial time. In an adversarial environment, the adversary can carry out a number of attacks, each of which adds special costs called *harm* to the graph, equivalent to adding a new cost vector to the graph. For example, with one attack the adversary may jam a specific region of the communication network, forcing nearby nodes to use more battery power due to higher transmission power or a greater number of retransmissions. In a physical environment, the adversary may place an obstacle on a road that slows the movement of vehicles over that road. When the adversary carries out multiple attacks, the cost vectors for each attack are summed and used to compute the total harm suffered by the agents in addition to the neutral environmental costs.

This interaction is naturally modeled using the theoretical framework of game theory. The harm depends on both the flows used by the agents and the attacks carried out by the adversary, and so the agents must reason about what attacks the adversary will carry out, even as the adversary contemplates what flows the agents will use. Assuming that both sides are rational, self-interested actors, game theory provides a rigorous framework for reasoning strategically about their behaviors. In particular, the game theoretic concept of Nash equilibrium is a solution in which both sides are playing optimally given the behavior

of the other side.

We model flow allocation in adversarial environments as a two-player game played on a directed graph. One player, the sender, represents the team of agents and chooses satisfying flows through the network for the source groups. The other player, the adversary, chooses a set of attacks from a set of possible attacks. The players choose their strategies without being able to observe the strategy chosen by the other player, but they are able to reason about what the other player would do based on common knowledge about the payoffs. We consider both zero-sum games, where the players receive payoffs that are exactly opposite (based on the harm suffered by the agents), and non-zero-sum games where the payoff to the sender is based on the harm and the environmental costs, while the payoff to the adversary is based on the harm and costs of playing each attack.

These games are challenging to solve using traditional game theory because the players have very large strategy spaces: the continuous space of all satisfying flows for the sender, and the exponential space of subsets of possible attacks based on the maximum number of attacks that can be carried out simultaneously. However, we show how the structure of the games can be used to efficiently find equilibrium solutions in polynomial time. We also show how various types of uncertainty about payoffs and adversary capabilities can be modeled as Bayesian games and extend our algorithm to solving these.

Finally, we consider games where the sender must commit to a strategy that can be observed by the adversary. Although the Nash equilibrium strategies can be used, we show that this may lead to very bad payoffs for the sender due to indifference on the part of the adversary. In a security setting, the sender cannot leave the difference between success and disaster up to the whims of the adversary. Instead, we show how the sender can deviate by an arbitrarily small amount from an equilibrium strategy in order to induce even a worst-case adversary to play a strategy that results in the most favorable expected payoff for the sender.

1.5 Contributions

This thesis makes the following contributions:

- a formulation of the flow allocation problem in capacitated environments and a proof that it is strongly NP-hard (Chapter 2);
- a pseudo-polynomial time algorithm for flow allocation on trees (Chapter 2);
- a proof that flow allocation on general graphs where flows cannot be split is strongly NP-hard to solve optimally and cannot be meaningfully approximated in polynomial

time unless P = NP (Chapter 2);

- a proof that network augmentation is NP-hard to solve optimally, and algorithms for solving it heuristically (Chapter 3);
- a linear programming formulation for finding equilibrium strategies in zero-sum flow allocation games (Chapter 4);
- a linear programming formulation for finding equilibrium strategies in non-zero-sum flow allocation games with costs (Chapter 5);
- a new equilibrium refinement, inducible Stackelberg equilibrium, for non-zero-sum flow allocation games with commitment, and an algorithm for computing the optimal inducible Stackelberg equilibrium (Chapter 5).

Chapter 2

Flow Allocation in Capacitated Environments

We begin by considering the problem of allocating flows in a capacitated environment. Communication networks are natural examples of these kinds of environments, as many multiagent systems use wireless networks that are tightly constrained in the amount of data that can be sent between agents. Despite these limitations, groups of agents must often transmit streams of data to a single location in the network as input to a computational task. For example, locating, recognizing, and tracking an object with a sensor network requires fusing data streams from multiple sensor, while plan monitoring requires access to status information from agents in the subteam executing the plan. Because of capacity constraints, it may not be possible for all tasks to be executed. The fundamental flow allocation problem is to choose the task locations and flows so that the maximum number of tasks can be performed. We formalize this problem next.

2.1 Flow Allocation

In this section we formalize the flow allocation problem. A problem instance has three parts: a graph, sink nodes, and groups of source nodes. These correspond to the environment, task execution locations, and agent subteams.

The environment is represented as a directed graph $G = (V, E)$ where V is the set of nodes and E is the set of edges; following convention we refer to such a graph as a *network*. The number of nodes and edges are denoted by $n = |V|$ and $m = |E|$, respectively. Each edge $e \in E$ has a weight $w(e) \in [0, +\infty]$ representing the capacity of that edge. The

Symbol	Description
$b(s), b_s$	source requirement of $s \in S$
E	set of edges
f_e^i	the amount of flow on edge $e \in E$ for group S_i
$G = (V, E)$	directed graph of the environment
K	number of groups
m	number of edges
n	number of nodes
$p(v)$	parent of node v
$S \subset V$	set of source nodes
$\mathfrak{S} = \{S_1, \ldots, S_K\}$	set of groups partitioning S
$t \in V \setminus S$	sink node
$Tr = (V, E, t, w)$	TREE-IN input tree with nodes E, edges E, root $t \in V$, and edge capacities w
V	set of nodes
$w(e)$	capacity of edge $e \in E$

Table 2.1: Summary of notation for Chapter 2

capacity of an edge may either be a non-negative real number (in which case the edge has limited capacity), or positive infinity (in which case the edge has unlimited capacity). As we will see later the inclusion of positive infinity as a value for $w(e)$ is convenient but not necessary as any in any instance a suitable large finite value could be substituted instead.

The set of sink nodes $T \subset V \setminus S$ represent the possible locations of task execution in the environment. With physical tasks, each sink represents the location of a task in the physical environment. For example, in the disaster response domain a sink may represent a burning building. With computational tasks, each group is assumed to have an implicit task, and the sinks represent processing nodes where the computation can be performed. For example, in a distributed sensor network each group corresponds to a set of sensor nodes that require data fusion of their sensor readings, and sinks correspond to base stations where this fusion can occur. We assume without loss of generality that sink nodes have no outgoing edges; if there is a sink node t with outgoing edges we can add a new sink node t', add an edge from t to t' with unlimited capacity, and remove t from T. An example of this process is shown in Figure 2.1.

The source nodes $S \subset V$ represent the agents in the environment. The source nodes are partitioned into K groups denoted $\mathfrak{S} = \{S_1, \ldots, S_K\}$ reflecting the organization of the

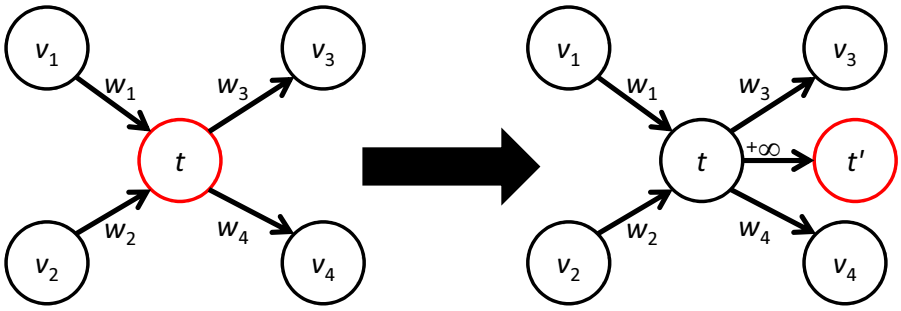


Figure 2.1: Transforming a network with outgoing edges from a sink node to one without outgoing edges from that sink node. A new sink node is added with an edge with unlimited capacity from the old sink node. This adds at most $|T|$ additional nodes and edges.

agents into non-overlapping subteams, with each source node in exactly one group. The group index $group(s)$ of a source node $s \in S$ is the index i such that $s \in S_i$.

The *flow requirement* is a function $b : S \to [0, +\infty)$ mapping source nodes to the amount of flow that must be sent from that source node to a sink. Because all source nodes in a group must send their flow to the same sink, it is often convenient to index the flow requirements both by the source node and the source group, denoted by $b_s^i = b(s)$ for $s \in S_i$. When convenient we may extend the domain of b to all nodes, not just those in S, with the understanding that $b(v) = 0$ for all $v \notin S$ and similarly $b_v^i = 0$ for all $v \notin S_i$.

A solution to the flow allocation problem has three components: a set of groups for which flow has been allocated, the assignment of sinks to these groups, and the flows that have been allocated. We let $\mathcal{X} \subseteq \mathfrak{S}$ denote the set of groups for which flow has been allocated. For each $S_i \in \mathcal{X}$ we let $x_i \in T$ denote the sink assigned to group S_i.

A *multicommodity network flow* for $\mathcal{X}$ is a function $f : E \to [0, +\infty]^{|\mathcal{X}|}$ that maps edges to a non-negative amount of flow on that edge for each group in $\mathcal{X}$. For convenience we usually denote the flow for the ith group on an edge $e = (u, v) \in E$ by f_e^i or f_{uv}^i, and in cases where there is a single group (i.e., $\mathfrak{S} = \{S\}$) we sometimes omit the superscript. A flow must satisfy the flow capacity and flow conservation constraints:

$$\sum_{S_i \in \mathcal{X}} f_e^i \leq w(e) \qquad \forall e \in E \qquad \text{(flow capacity)} \qquad (2.1)$$

$$b_v^i + \sum_{(u,v) \in E} f_{uv}^i = \sum_{(v,u) \in E} f_{uv}^i \qquad \forall v \in V \setminus T, S_i \in \mathcal{X} \qquad \text{(flow conservation)} \qquad (2.2)$$

The capacity requirement constraint in Equation (2.1) requires that the total flow for all

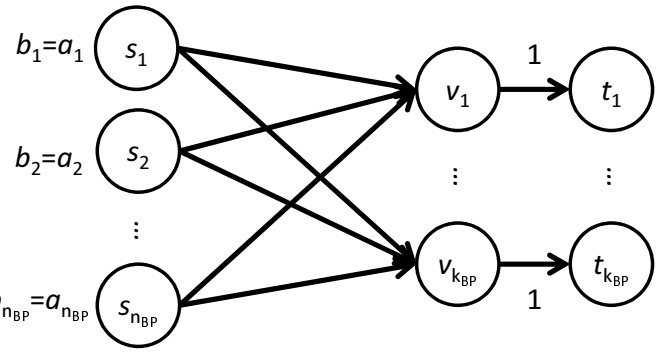


Figure 2.2: MaxSG-Dec is strongly NP-complete: Reduction from the Bin Packing decision problem.

groups on each edge must be no greater than the capacity of that edge. The flow conservation constraint in Equation (2.2) requires that the amount of flow of each type that exits a node is equal to the sum of the amount of flow of that type originating at that node (i.e., the flow requirement) and the amount of flow of that type entering the node, for all nodes other than the sink. From these two constraints it is clear that no edge can receive a total amount of flow that exceeds $B = \sum_{s \in S} b_s$, and so the weights $w(e)$ could be limited to the interval $[0, B]$.

The flow for each group $S_i \in \mathcal{X}$ should be sent to the sink x_i assigned to it. This is expressed in the sink flow constraint:

$$\sum_{(u,x_i) \in E} f^i_{ux_i} = \sum_{s \in S_i} b^i_s \qquad \forall S_i \in \mathcal{X} \qquad \text{(sink flow)} \qquad (2.3)$$

If a flow satisfies the sink flow constraint in Equation (2.3) we say that the flow is a *satisfying flow* for $\mathcal{X}$ and that the groups in $\mathcal{X}$ are *satisfied*. Satisfied groups correspond to agent subteams that are able to proceed with task execution. A natural problem is to enable as many subteams as possible to execute tasks. This is the problem of maximizing the number of satisfied groups, MaxSG.

Problem 1. Maximum Satisfied Groups (MaxSG). Given graph $G = (V, E)$ with capacities w, source groups $\mathfrak{S}$, source requirements b, and sinks T, find $\mathcal{X} \subseteq \mathfrak{S}$, sink assignment x and satisfying flow f for $\mathcal{X}$ such that $|\mathcal{X}|$ is maximized.

The decision version of this problem, MaxSG-Dec, is to decide whether it is possible to allocate flows to satisfy at least k groups.

Theorem 1. *MaxSG-Dec is strongly NP-complete.*

Proof. To show that MaxSG-Dec is in NP we observe we can verify in polynomial time if $|\mathcal{X}| \geq k$ and f is a satisfying flow for $\mathcal{X}$.

We prove NP-hardness by reduction from Bin Packing, which is known to be strongly NP-complete [21]. The Bin Packing decision problem is the NP-complete problem of deciding if a set of n_{BP} items of different size can be placed into k_{BP} bins. Formally, it is the problem of deciding if a set of numbers $\{a_1, \ldots, a_{n_{BP}}\}$ with $a_i \in (0, 1]$ can be partitioned into k_{BP} sets $Bin_1, \ldots, Bin_{k_{BP}}$ such that $\sum_{a \in Bin_i} \leq 1$ for $1 \leq i \leq k_{BP}$.

From a Bin Packing instance we construct an instance of MaxSG-Dec, as illustrated in Figure 2.2. First, create n_{BP} source nodes $s_1, \ldots, s_{n_{BP}}$ and set the flow requirements $b_i = a_i$. Place each source node in its own group. Next create k_{BP} nodes $v_1, \ldots, v_{k_{BP}}$ and add edges from each source node to each of these newly created nodes. Set the capacity of these edges to positive infinity. Add k_{BP} sink nodes, $t_1, \ldots, t_{k_{BP}}$ and add an edge with unit capacity between v_i and t_i for $1 \leq i \leq k_{BP}$. Finally set $k = n_{BP}$ so that the decision is whether it is possible to allocate flow for all of the groups.

Suppose that there is a flow satisfying all n_{BP} groups. Because each group is a singleton, this means that the flow from each source node can be sent to sinks. Let $Y_i \subseteq S$ be the set of all source nodes sending flow to sink t_i, that is, $Y_i = \{s_j \in S : x_j = t_i\}$. From the flow capacity constraint for edge (v_i, t_i), it follows that

$$\sum_{s_j \in Y_i} a_j \leq 1 \qquad \forall i \in [1..k_{BP}]$$

Because all groups are satisfied and each group is a singleton, it follows that the $\{Y_i\}$ partition S into at most k_{BP} sets. Thus there is a bin packing using at most k_{BP} bins.

Conversely, suppose that there is a bin packing using at most k_{BP} bins. For each $a_j \in Bin_i$ we set $x_j = t_i$ and $f^j_{s_j v_i} = f^i_{v_i t_i} = a_j$ (and 0 for other edges). By the bin packing constraints we know that the flow capacity constraints on the edges (v_i, t_i) must be satisfied and hence all n_{BP} groups are satisfied.

The answer to a Bin Packing instance is YES if and only if the answer to the MaxSG-Dec instance is YES, so MaxSG is strongly NP-hard, and therefore strongly NP-complete. □

We solve MaxSG by formulating it as a mixed integer linear program (MILP) and using existing algorithms for solving MILPs to compute a solution. Solving a MILP is known to be NP-hard, but this is acceptable given Theorem 1. Our formulation for MaxSG is MILP 1. We must specify the flow capacity, flow conservation, and sink flow constraints as a set of constraints that are linear in the variables being solved for. At first glance, it may appear that Equations (2.1) – (2.3) are of this form as they are linear in terms of f.

MILP 1 MILP formulation of MaxSG.

$$\underset{f,X}{\text{Maximize}} \sum_{i=1}^{K} \sum_{t \in T} X_{it} \tag{2.4}$$

subject to:

$$b_v^i \sum_{t \in T} X_{it} + \sum_{(u,v) \in E} f_{uv}^i = \sum_{(v,u) \in E} f_{uv}^i \qquad \forall v \in V \setminus T, i \in [1..K] \tag{2.5}$$

$$\sum_{i=1}^{K} f_e^i \leq w(e) \qquad \forall e \in E \tag{2.6}$$

$$\sum_{(u,t) \in E} f_{ut}^i = X_{it} \sum_{s \in S_i} b_s^i \qquad \forall t \in T, i \in [1..K] \tag{2.7}$$

$$\sum_{t \in T} X_{it} \leq 1 \qquad \forall i \in [1..K] \tag{2.8}$$

$$X_{it} = 0 \qquad \forall i \in [1..K], t \in T \tag{2.9}$$

$$f_e^i \geq 0 \qquad \forall e \in E, i \in [1..K] \tag{2.10}$$

$$X_{it} \in \{0, 1\} \qquad \forall i \in [1..K], t \in T \tag{2.11}$$

However, the output $\mathcal{X}$ is also used in all three: to determine the sum in the flow capacity constraint, and to determine the presence or absence of constraints for different values of i in the flow conservation and sink flow constraints. We linearize these constraints through the addition of integer indicator variables X_{it} for $1 \leq i \leq K$ and $t \in T$. Each variable can take the value 1 or 0, with $X_{it} = 1$ if t is the sink assigned to S_i (that is, if $x_i = t$) and $X_{it} = 0$ otherwise. Because each group is assigned at most one sink, it follows that at most one X_{it} for each i is non-zero. This is expressed by the constraint in Equation (2.8). If the sum equals 0, then no sink has been assigned to group S_i and so $S_i \notin \mathcal{X}$. In this way the integer collectively represent $\mathcal{X}$ in addition to the sink assignment x. As a result, the objective of maximizing $|\mathcal{X}|$ is equivalent to maximizing the sum of the X_{it} variables, which is the objective of MILP 1 as shown by Equation (2.4).

Flow conservation is expressed as Equation (2.5), which requires that flow originates at v for group S_i if and only if a sink has been assigned to S_i. As a result, for all $e \in E$, it follows that $f_e^i > 0$ if and only if a sink has been assigned to S_i, and so we can express the flow capacity constraint by summing over all possible groups as shown in Equation (2.6), knowing that there will only be a positive amount of flow for groups in $\mathcal{X}$. Finally, the sink flow constraint is captured by Equation (2.7), which requires that the total amount of flow for group S_i entering a sink t must be $\sum_{s \in S_i} b_s^i$ if t is the sink assigned to S_i and 0 otherwise.

2.1.1 Alternative Capacity Models

We consider flows that are limited by capacities on the edges. In this subsection we describe how to accommodate some other types of capacities.

Node Capacities

With node capacities the amount of flow that can pass through each node $v \in V$ is limited to a maximum amount $w(v) > 0$. In communication networks this arises when the ability of a node to process communication is limited. One way this can arise is when the team includes humans who must manually relay the information.

A graph $G = (V, E)$ with node capacities can be transformed into a new graph $G' = (V', E')$ with an additional $|V|$ nodes and edges. This construction is illustrated in Figure 2.3. For each node $v \in V$ we add two nodes v_{in} and v_{out} to V', and an edge (v_{in}, v_{out}) to E' with edge capacity $w(v_{in}, v_{out}) = w(v)$. This edge represents the limitation in the node v's ability to process flow. To preserve the overall topology of the original network,

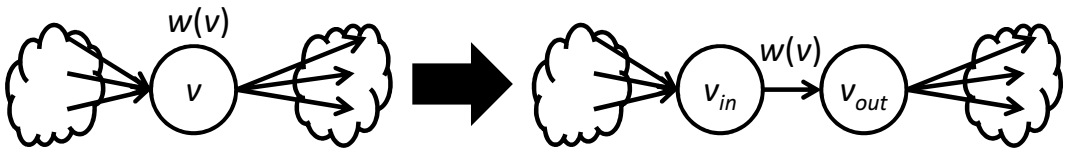


Figure 2.3: Transforming a network with node capacities to one with edge capacities.

for each edge $(u, v) \in E$ we add an edge (u_{out}, v_{in}) to E'.

In the MILP formulation we can more easily incorporate node capacities by adding a new constraint:

$$\sum_{(u,v)\in E} f_{uv}^{i} \leq w(v) \qquad \forall v \in V \qquad \text{(node capacities)}.$$

Wireless Interference

Many multiagent teams utilize wireless networks that share the same channel for communication. In such cases, individual capacity limitations on each link are often far less important than the effect of wireless interference. Most nodes use omnidirectional antennas and the transmissions from a node are heard by all nearby nodes, causing interference. A simple way to model this to a first approximation in the MILP formulation uses adjacency in G as an indication of which nodes can interfere, on the premise that if the transmission from a node can be used to communicate with another node, it can also interfere with that node. More sophisticated approaches may use a neighbors-of-neighbors model to reflect the rule-of-thumb that the interference range is generally twice the transmission range [65], or conflict graphs that explicitly describing which links interfere [31].

We assume that transmissions use a shared channel and the edge capacities reflect full utilization of that channel in the absence of other transmissions. When a node $v \in V$ transmits on multiple edges, it divides time transmitting to each recipient, with the fraction of time equal to $f_{vu}/w(v, u)$. This leads to the following constraint on outgoing flow:

$$\sum_{(v,u)\in E} \frac{f_{vu}}{w(v,u)} \leq 1 \qquad \forall v \in V$$

To receive a transmission, v must have an open time slot to receive the transmission. However, due to interference time slots are filled with all transmissions from neighbors

of v, not just those intended for v. We first consider the available time slots without considering interference. We add a binary decision variable Z_v for each $v \in V$ to indicate whether v can receive flow. If $Z_v = 1$ then flow is allowed to be sent to v; if $Z_v = 0$ then no flow can be sent to v. Thus we get the constraint

$$\sum_{(u,v)\in E} \frac{f_{uv}}{w(u,v)} \leq Z_v \qquad \forall v \in V.$$

Next we consider the effect of interference. If flow is sent to v, the total time slots must be used between received transmissions and interference. If flow is not sent to v we don't care how much time slots are suffer from interference. This is achieved by using a suitably large constant M in the following constraint:

$$\sum_{(u,v)\in E} \sum_{(u,v')\in E} \frac{f_{uv'}}{w(u,v')} \leq 1 + M(1 - Z_v) \qquad \forall v \in V.$$

When $Z_v = 1$, the right-hand side becomes 1, so at most all of the time slots can be used. When $Z_v = 0$ the right-hand side becomes $M + 1$, removing the limit on the amount of interference for suitably large M. However, v cannot receive any flow due to the previous constraint.

2.2 Flow Allocation on Trees

We turn now from environments modeled as general directed graphs with multiple sinks to environments with a more limited topology and a single sink. In communication networks, simple network structures, and trees in particular, are often favored for reasons of efficiency and scalability. Trees greatly simplify routing, while also arising naturally as a result of power-conserving mechanisms such as hierarchical routing [28, 29], data aggregation [1], and topology control [49]. If flow allocation for computational tasks is done in these kinds of networks, the functional network topology used as input will be a tree.

In this section we consider a special case, the TREE-IN case, in which $G = Tr = (V, E, t)$ is a directed tree with a single sink t at the root and all edges directed toward the root. In addition, we assume that the source nodes are each in their own group, so that $\mathfrak{S} = \{\{s\} : s \in S\}$. Because of this assumption, in this section we find the set of source nodes $S' \subseteq S$ whose flows can be allocated, with the understanding that this is equivalent to finding $\mathcal{X} = \{\{s\} : s \in S'\}$.

We denote the subtree rooted at a node v (including the node v itself) by $Tr_v = (V_v, E_v)$. We also use $p(v)$ to denote the parent of any nodes $v \in V$ other than the root. Without loss of generality we can assume that there are no leaf nodes in Tr that are not source nodes; if there are, we can remove them without affecting the problem because they cannot be on a flow path from any of the sources to the sink. Furthermore, we can assume that all source nodes are leaves in Tr; for an interior source node s, we can add a new leaf node s' with edge (s', s), designate s' as a source node with $b_{s'} = b_s$, and remove s from S. Together these assumptions mean that we can assume a node is a leaf if and only if it is a source node.

The key characteristic of the tree topology is that there is a unique path from v to the root t for any $v \in V$. Because of this, the flow conservation constraint requires that all of the flow that originates in a subtree Tr_v (for $v \neq t$) must be transmitted over the edge $(v, p(v))$. If S' is the set of source nodes whose groups are satisfied, it follows that $V_v \cap S'$ are the satisfied source nodes in the subtree rooted at v, and so $\sum_{s' \in V_v \cap S'} b_{s'}$ is the total amount of flow on edge $(v, p(v))$. For the TREE-IN case, the flow capacity constraint is equivalent to requiring that that amount of flow is no greater than $w(v, p(v))$ for all v other than the root.

The following formalizes the TREE-IN case of the MaxSG problem.

Problem 2. (TREE-IN case of MaxSG.) Given a directed tree $Tr = (V, E, t)$ with root $t \in V$, integral edge capacities w, and source nodes S with integral source requirements b, find $S' \subseteq S$ such that

1. $|S'|$ is maximized; and
2. $\sum_{s' \in V_v \cap S'} b_{s'} \leq w(v, p(v))$ for all $v \in V \backslash \{t\}$.

The first condition is equivalent to maximizing the number of satisfied source groups as each source node is in its own group. The second condition is equivalent to the flow capacity constraint, as described above.

Recall from the proof Theorem 1 that a Bin Packing instance was reduced to an instance of MaxSG where each source node was in its own group. Thus, MaxSG is still strongly NP-hard even when restricted to source groups that are singletons, and so there are no known pseudopolynomial-time algorithms for solving these instances. A natural question is whether this remains true in the TREE-IN case, where there is a single sink and the graph is a tree.

The answer is no. Algorithm 1 is a multi-layer dynamic programming algorithm that solves the TREE-IN case optimally in pseudo-polynomial time. It proceeds by computing two functions, one for the value of an optimal solution and one for an optimal solution itself, in a bottom-up fashion. The first function is $F_v(y)$, the maximum number of source

nodes in the subtree rooted at v (including v itself) that can have their flow requirements satisfied if the total amount of flow sent to v is at most y. The second function is $S_v(y)$, a maximum set of source nodes in the subtree rooted at v (including v itself) that can have their flow requirements satisfied if the total amount of flow sent to v is at most y.

For each $v \in V$, we must compute $F_v(y)$ and $S_v(y)$ for values of $y \in [0..B_v]$, where $B_v = \sum_{s \in S \cap T_v} b(s)$, the sum of flow requirements in the subtree rooted at v. As we move up the tree, we compute $F_v(y)$ and $S_v(y)$ based on the values for the children of v. The optimal substructure that we exploit is that $F_v(y)$ is determined exactly by the values $F_u(z)$ for its children u, and that we can compute $F_v(y)$ sequentially by iterating over the children of v.

We begin Algorithm 1 by initializing B_v for all v (line 1). These values will be used later in the algorithm. In line 2 we then order the nodes according to their depth-first post order. This guarantees that all children of a node v will be processed before v is processed, so that we can recursively use the F values for the children in computing the F value for v. We then iterate through the nodes to compute the F_v and S_v values (lines 3–22). The base case applies when v is a source (i.e., leaf) node. In this case we can satisfy the source node if and only if we use flow at least equal to b_v. Thus, if $y < b$ then $F_v(y) = 0$ and $S_v(y) = \emptyset$, while if $y = b_v$ then $F_v(y) = 1$ and $S_v(y) = \{v\}$ (lines 5–6). Note that F_v and and S_v do not depend on the capacity of the edge $(v, p(v))$; this constraint will be taken into account when we compute the values for interior nodes.

For an interior node v, we recursively compute $F_v(y)$ based on the values $F_u(z)$ for children u of v. If y total flow is allowed to be sent to v, this flow must come from the children of v and hence we would like search the space of all possible ways of dividing the y flow for v among the children of v. However, naively enumerating these takes exponential time. Instead, we use another level of dynamic programming. The optimal substructure that we exploit is that the optimal way to divide y flow among l children must be some combination of optimally dividing between z flow among the first $l-1$ children and reserving the remaining $y - z$ flow for the lth child, for some $z \in [0..y]$.

In lines 13–21 we iterate through the children. At the beginning of iteration l, we have $F_v^{l-1}(y)$, the F_v values computed by considering the first $l-1$ children, and the corresponding S_v values; we compute the values for the lth iteration for all possible y (the inner loop on lines 14–20). For a given value of y, we compute $F_v^l(y)$ by exhaustively considering all ways to divide the y units of flow into z units of flow for the first $l-1$ children and the $y - z$ units of flow for the lth child (the loop from lines 16 – 20); we choose the division that maximizes the number of satisfied groups. It is at this stage that we check for capacity constraints on line 17. the amount of flow that can be obtained from the child u is bounded by the capacity of the edge $w(u, v)$ and B_u.

Algorithm 1 TREE-IN

Input: $Tr = (V, E, t)$; edge capacities w; source nodes $S \subset V$ with b_s for all $s \in S$.
Output: Maximum $S' \subseteq S$ that satisfies all flow constraints.

Set B_v to be the sum of flow requirements in the subtree rooted at v, for all $v \in V$
Order V by the depth-first post order
for all $v \in V$ **do**
  if v is a source node **then**
    $F_v(y) \leftarrow 0$ for $0 \leq y < b_v$
    $S_v(y) \leftarrow \emptyset$ for $0 \leq y < b_v$
    $F_v(b_v) \leftarrow 1$
    $S_v(b_v) \leftarrow \{v\}$
  else if v is an interior node **then**
    $F_v^0(y) \leftarrow 0$ for $0 \leq y \leq B_v$
    $S_v^0(y) \leftarrow \emptyset$ for $0 \leq y \leq B_v$
    $l \leftarrow 1$
    for all child u of v **do**
      for $y \leftarrow 0$ **to** B_v **do**
        $F_v^l(y) \leftarrow 0$
        for $z \leftarrow 0$ **to** y **do**
          $\Delta \leftarrow \min(y - z, w(u, v), B_u)$
          if $F_v^l(y) < F_v^{l-1}(z) + F_u(\Delta)$ **then**
            $F_v^l(y) \leftarrow F_v^{l-1}(z) + F_u(\Delta)$
            $S_v^l(y) \leftarrow S_v^{l-1}(z) \cup S_u(\Delta)$
      $l \leftarrow l + 1$
    $F_v(y) \leftarrow F_v^{l-1}(y)$ for $0 \leq y \leq B_v$
    $S_v(y) \leftarrow S_v^{l-1}(y)$ for $0 \leq y \leq B_v$
return $S_t(B_t)$

After we have looped through all children, we take F_v and S_v to be the values of we computed in the final iteration (lines 22 and 23), as these are the values after considering all of v's children. Note that it is important that we compute $F_v(y)$ and $S_v(y)$ for all possible values of y. This is because we will need to use them when considering the parent of v, which must consider all possible ways to divide its flow between its children (i.e., v and v's siblings).

The running time of Algorithm 1 is pseudo-polynomial. It is clear that we can compute lines 1 and 2 in $O(n+m)$ time. Observe that $B_t = \max_v B_v$. Within the main body of the algorithm, the innermost loop (lines 16–20) takes at most $O(B_t)$ time and it is repeated at most $O(B_t)$ time in the loop from lines 14 – 20. As this loop in turn is repeated for each child of interior node, the innermost loop is repeated at most $O(n)$ times in total, for a time of $O(nB_t^2)$. The rest of the outermost loop body in lines 4–23 take at most $O(B_t)$ time, so the total running time is $O(nB_t^2 + m)$, which is pseudo-polynomial in the size input due to its polynomial dependence on the magnitude of B_t.

2.3 Unsplittable Flow Allocation

In this section we consider the case where the flows cannot be divided over multiple paths as they could be in the previous problem formulations. Instead, the flow from each source node must be routed along a single path to the sink node. This assumption may be the case for some communication networks and is also applicable when the flows represent the movement of physical agents (such as a robot) that cannot be divided. We consider consider general directed graph inputs and seek to maximize the number of source groups that can fully transmit their required flows to the sink.

Problem 3. (MaxSG-UF) Given a directed graph $G = (V, E)$ with integral edge capacities $w(e) > 0$ for all $e \in E$, source groups $\mathfrak{S}$, integral source requirements b, and sink nodes T, find $\mathcal{X} \subseteq \mathfrak{S}$, sink assignment x, and satisfying unsplittable flow f for $\mathcal{X}$ such that $|\mathcal{X}|$ is maximized.

The decision problem associated with this optimization problem, MaxSG-UF-Dec, is to decide whether a MaxSG-UF instance has a solution with $|\mathcal{X}| \geq k$.

Theorem 2. *The MaxSG-UF-Dec decision problem is strongly NP-complete.*

Proof. Given a flow f from the source nodes to sinks, it is straightforward to check in polynomial time whether they meet the flow requirements for at least k groups, so this is clearly in NP.

We prove strong NP-hardness by reduction from 3-Partition, which is known to be strongly

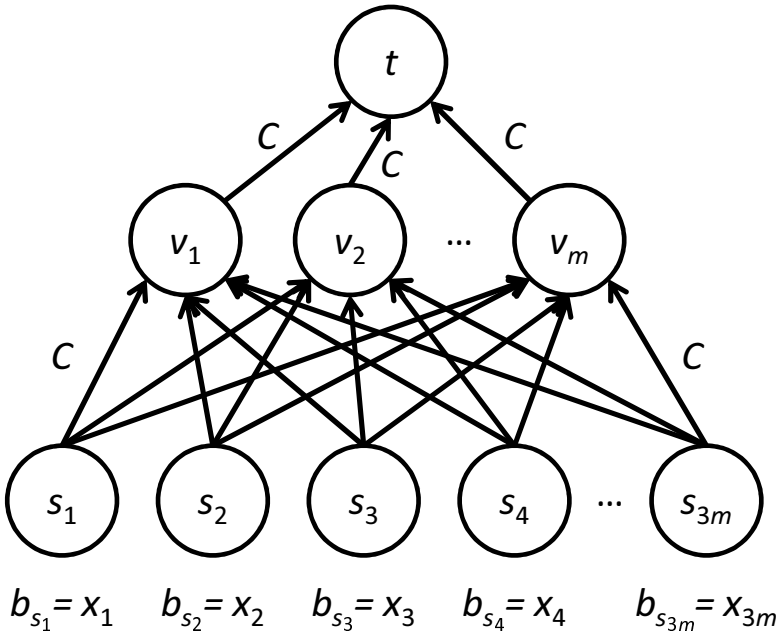


Figure 2.4: MaxSG-UF-Dec is strongly NP-complete: Reduction from 3-Partition.

NP-complete [21]. The 3-Partition decision problem is as follows: Given a set $A = \{a_1, \ldots, a_{3m}\}$ of $3m$ positive integers where $\sum_{a \in A} a = mC$, and $C/4 < a < C/2$ for all $a \in A$, decide whether A be partitioned into m subsets $A_1, \ldots, A_m$ such that $\sum_{a \in A_i} a = C$ for all A_i.

Given a 3-Partition instance, we construct a MaxSG-UF-Dec instance, as illustrated in Figure 2.4. We create a sink node t, then add m intermediate nodes $v_1, \ldots, v_m$ and connect each one to t with edge (v_i, t) with capacity $w(v_i, t) = C$. We then create $3m$ source nodes $s_1, \ldots, s_{3m}$ and for all source nodes s_i and intermediate nodes v_j, we add an edge (s_i, v_j) with capacity C. We set the source requirements $b_{s_i} = x_i$ for $1 \leq i \leq 3m$, and ask whether it is possible to satisfy all $k = 3m$ source groups.

Suppose that the 3-Partition decision is YES. Then for each $A_i = \{a_{k_{i1}}, a_{k_{i2}}, a_{k_{i3}}\}$ we have that $a_{k_{i1}} + a_{k_{i2}} + a_{k_{i3}} = C$ and therefore we can route the flow from $s_{k_{i1}}$, $s_{k_{i2}}$, and $s_{k_{i3}}$ to t via node v_i. Because the A_i cover A, it is possible to satisfy the source requirements for all source nodes. Because the A_i are pairwise disjoint, each $a_j \in A$ is in exactly one A_i and hence it is possible for $s_j \in S$ to send its flow only to v_i. Thus the flow for each source node can be routed on a single path to t, and so the MaxSG-UF-Dec decision is YES.

Conversely assume that the MaxSG-UF-Dec decision is YES. From the construction of

capacities and source requirements, each intermediate node must receive flow from exactly three children $\{s_{k_{i1}}, s_{k_{i2}}, s_{k_{i3}}\}$, and their combined flow must equal C. Furthermore, no source node sends flow to more than one intermediate node. Hence the sets $A_i = \{a_{k_{i1}}, a_{k_{i2}}, a_{k_{i3}}\}$ satisfy the 3-Partition requirements, and so the 3-Partition decision is YES.

Thus the 3-Partition decision is YES if and only the constructed MaxSG-UF-Dec decision is YES, and so MaxSG-UF-Dec is strongly NP-complete. □

By Theorem 2 there is no known pseudopolynomial-time algorithm for solving MaxSG-UF optimally. We next show that this problem is hard to even approximate.

Theorem 3. *It is NP-hard to approximate MaxSG-UF within a factor of $K^{1-\epsilon}$ for any $\epsilon > 0$ where m is the number of edges of the graph in the MaxSG-UF problem.*

Proof. The proof is by reduction from the NP-hard problem 2DIRPATH [26]. An instance of 2DIRPATHis a directed graph $H = (V_H, E_H)$ and distinct nodes $x^1, x^2, y^1, y^2 \in V_H$. The problem is to decide whether there exist directed paths from x^1 to y^1 and x^2 to y^2 such that the paths share no common edges (i.e., they are edge-disjoint).

The reduction is a modified version of the one used in [26]. Given any $\epsilon > 0$, we construct a directed graph G from H. Figure 2.5 provides an illustration of the construction.

Let $N = |V_H|^{\lceil 1/\epsilon \rceil}$. We first create N source nodes s_i for $i \in [1..N]$, with source requirements $b_{s_i} = i$. We place each source node in its own group so that $\mathfrak{S} = \{\{s\} : s \in S\}$. We then create N sink nodes t_i for $i \in [1..N]$. Place source nodes from right to left on horizontal axis, s_i at position $n - i + 1$ and sink nodes from bottom to top on vertical axis, t_i at position i, where $1 \leq i \leq N$. As in Figure 2.5, we connect s_i to t_i by draw a line from s_i, then going up until a node $d_{i,i}$ at height i, then going left until connecting t_i.

For each pair $1 \leq j < i \leq N$ we create a modified version of H that we denote H_{ij} by adding a capacity of i to each edge. The nodes corresponding to x^1, x^2, y^1, y^2 in H are denoted $x^1_{ij}, x^2_{ij}, y^1_{ij}, y^2_{ij}$, respectively, in H_{ij}. Suppose we wish to send $j' < j$ units of unsplittable flow from x^1_{ij} to y^1_{ij} and i units of unsplittable flow from x^2_{ij} to y^2_{ij}. This is possible if and only if there exist edge-disjoint paths in H from x^1 to y^1 and x^2 to y^2, because each edge e in H_{ij} can only carry flow from one of x^1_{ij} or x^2_{ij} due to the capacity constraint $w(e) = 1$, and the flows from each of x^1_{ij} and x^2_{ij} cannot split and must therefore follow single paths.

Returning now to our construction of G, we place the graph H_{ij} at the intersection of the path from s_i to t_i and the path from s_j to t_j where $1 \leq j < i \leq n$. We connect the path from s_i to x^2_{ij} and y^2_{ij} to the path to d_i. We also connect the path from d_j to x^1_{ij} and y^1_{ij} to the path to t_j. Formally this is achieved in the following way. For $i \in [1..N]$ we add edges

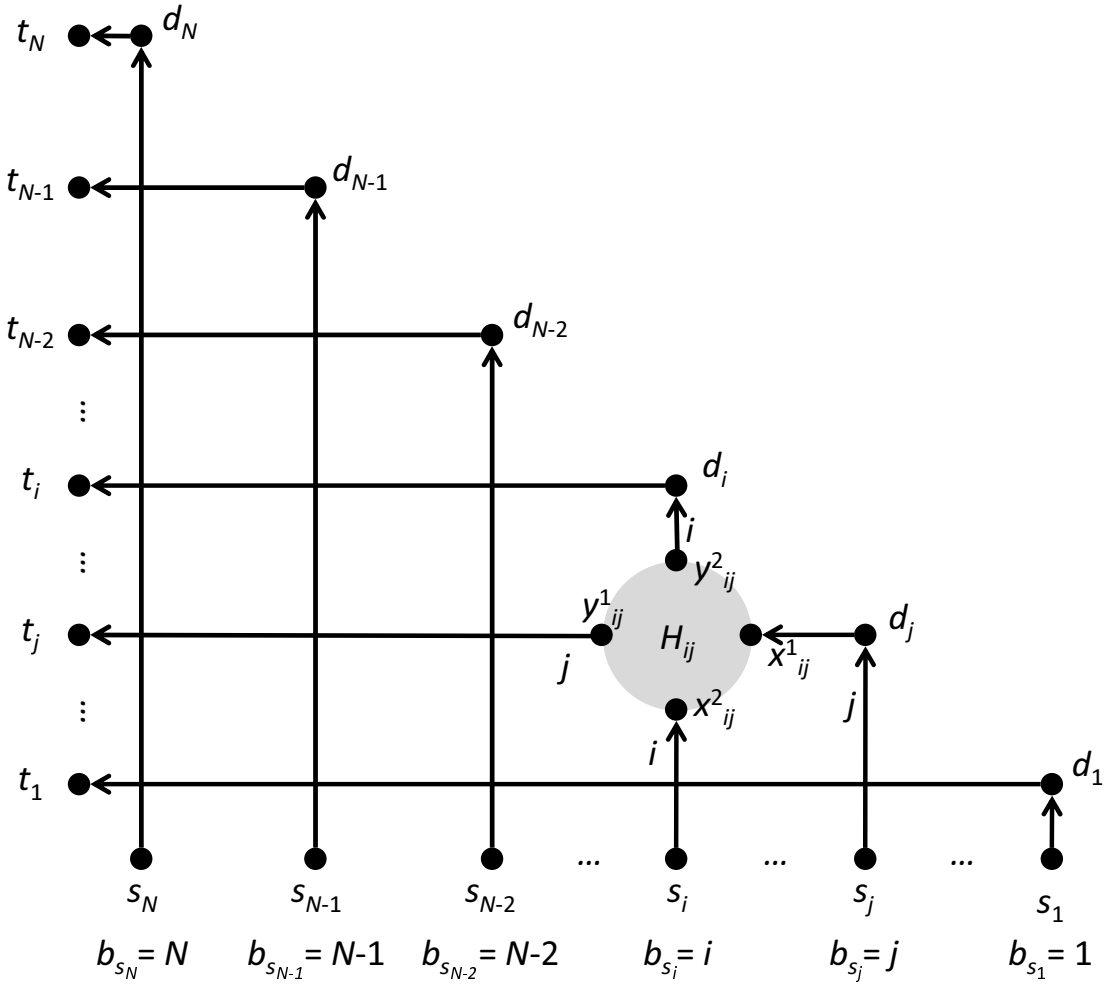


Figure 2.5: MaxSG-UF is hard to approximate: Reduction from 2DIRPATH.

(s_i, x^2_{i1}) and $(y^2_{i(i-1)}, d_i)$ with capacities i. For $j \in [1..N]$ we add edges $(d_j, x^1_{(j+1)j})$ and (y^1_{Nj}, t_j) with capacities j. For all $2 \leq j < i \leq N-1$, we add edges $(y^2_{i(j-1)}, x^2_{ij}$ with capacity i and $(y^1_{(i-1)j}, x^1_{ij})$ with capacity j.

CLAIM 1. If there exist edge-disjoint paths from x^1 to y^1 and x^2 to y^2 in H, then all source groups in G can be satisfied.

Proof of Claim 1. Suppose there exist such edge-disjoint paths in H. By our earlier observation, it follows that it is possible to route j units of flow from x^1_{ij} to y^1_{ij} and i units of flow from x^2_{ij} to y^2_{ij} in H_{ij} for all $1 \leq j < i \leq N$. By construction of the other edges of G, for each $s_i \in S$ it is therefore possible to send $b_{s_i} = i$ units of flow on the path from s_i to d_i by taking edges "up" in Figure 2.5, then from d_i to t_i by going "left." Since each s_i forms its own group in $\mathfrak{S}$, it follows that all source groups can be satisfied. □

CLAIM 2. If there do not exist edge-disjoint paths from x^1 to y^1 and x^2 to y^2 in H, then at most one source group in G can be satisfied.

Proof of Claim 2. We prove this claim by contradiction. Suppose that two source groups $\{s_i\}$ and $\{s_j\}$ can be satisfied, with $j < i$. Observe that by the construction of capacities, the flow from s_i must pass through d_i, and the flow from s_j must pass through d_j. Because $j < i$, it follows that the flow from s_j must "cross" the flow from s_i at some $H_{ij'}$ with $j \leq j' < i$. That is, there exists j' with $j \leq j' < i$ such that i units of flow from s_i enters $x^2_{ij'}$ and exits $y^2_{ij'}$ and j units of flow from s_j enters $x^1_{ij'}$ and exits $y^1_{ij'}$. By assumption H does not contain edge-disjoint paths from x^1 to y^1 and x^2 to y^2 and so by our previous observation, it is not possible to route the desired flows through $H_{ij'}$. But this means that it is not possible to satisfy both $\{s_i\}$ and $\{s_j\}$, which is a contradiction, thus completing our proof of the claim. □

By the above two claims, YES instances of 2DIRPATHare mapped to instances of MaxSG-UF where all N source groups can be satisfied, while NO instances are mapped to instances of MaxSG-UF where at most one source group can be satisfied. Suppose we had an approximation algorithm with a ratio $K^{1-\epsilon}$. Then given an instance of 2DIRPATH, we could construct an instance of MaxSG-UF with N source groups as described above in polynomial time. We know that it must be possible to either satisfy all the source groups, or satisfy at most one of them. We then run the approximation algorithm.

If all N source groups can be satisfied, the approximation ratio guarantees that we would be able to find a solution with at least $N/N^{1-\epsilon} = N^{\epsilon}$ groups satisfied. Because $N = |V|^{\lceil 1/\epsilon \rceil}$, we are thus guaranteed to find a solution with at least $|V|$ groups satisfied. By

assumption $|V| \geq 4$ (the starting and ending nodes in the 2DIRPATHmust all be distinct) so we can satisfy at least 4 groups.

If, on the other hand, at most one source group can be satisfied, the approximation algorithm will also only be able to find a solution that satisfies at most one group. Therefore, by seeing if the approximation algorithm can satisfy more than one group we will be able to decide 2DIRPATH. But 2DIRPATHis NP-hard, and so approximating MaxSG-UF within $K^{1-\epsilon}$ is also NP-hard. This completes the proof of Theorem 3. □

Theorem 3 essentially rules out any non-trivial approximation algorithm with a provable performance guarantee. The best that we can hope to do, unless $P = NP$, is to consider each group in isolation and try to find a satisfying flow, even when each group contains only a single source node. In that case, finding a satisfying flow reduces to finding a directed path from the source node s to a sink, using only edges with capacity $w(e) \geq b_s$.

Chapter 3

Network Augmentation and Flow Allocation

In this chapter we consider environments that the agents can modify by adding nodes and edges to improve connectivity and increase capacity, a process we term *network augmentation*. This is primarily motivated by the challenge of maintaining effective wireless communication networks for multiagent teams. Agents must often operate in noisy, obstacle-filled environments equipped with communication hardware that is size- and power-limited, leading to poor network performance. One way to remedy this is to supplement the agent-based network by deploying additional nodes to act as relays.

Agents can also affect physical environments, such as the road network in an urban disaster response domain. For example, police officers may be deployed to direct vehicular traffic, thereby reducing congestion and effectively increasing the road network's capacity to transport resources. In another example, robots may be deployed to restore edges in the transportation network by clearing away debris from obstructed roads.

The key problem in network augmentation is to determine where to best deploy the additional nodes. This partly depends on factors intrinsic to the environment: which agents will be in range if a relay is placed in a specific spot, for example, or what roads are currently obstructed but can be cleared. It also depends on the flows through the environment: low capacity only becomes a bottleneck if it is being used. But the choice of flows also depends on the environment, and hence on the choice of network augmentation. This highlights the interdependence of the network augmentation and flow allocation problems. Because of this interdependence, in this chapter we simultaneously optimize both the placement of supplemental nodes and the allocation of flows.

As in the previous chapter, we represent the environment as a directed graph with capacities on edges. We consider the *initial network* to be the graph of the environment without any supplemental nodes. We explicitly represent the ways in which the agents can augment the network by the *potential network*, which adds to the initial networks a set of nodes representing the locations where supplemental nodes can be deployed to, along with additional edges indicating the connectivity that would result if supplemental nodes were deployed to those locations. The potential locations are chosen a priori possibly by domain specialists or domain-specific algorithms (such as identifying feasible locations for airborne relays, or the locations obstructed roads), or as finite approximations (in the case of relays positioned in a continuous space).

In this chapter we formalize the problem of network augmentation and flow allocation, analyzing its complexity and providing algorithms for its solution. A summary of the notation used in this chapter is provided in Table 3.1. Portions of the work in this chapter was originally published in Okamoto and Sycara [48].

3.1 Network Augmentation

We know that the problem of network augmentation and flow allocation is NP-hard, because the flow allocation problem formulated in the previous chapter is a special case where the initial and potential networks are identical. However, in this section we consider a form of "pure" network augmentation that removes as many of the features of flow allocation as possible in order to focus on the issues inherent to the network augmentation side of the problem. We show that even this simplified pure network augmentation problem is NP-hard.

In pure network augmentation, we are given an initial network topology, a finite set of potential locations where supplemental nodes can be deployed, and groups of source nodes representing agents that need to communicate. The simplest case of network augmentation arises when all edges have unlimited (i.e., positive infinity) capacity and are symmetric. The problem is to deploy supplemental nodes to the potential locations in order to satisfy group requirements. In this case, group satisfaction reduces to connectivity, so a group of source nodes is satisfied if all the nodes in the group are connected to each other once the supplemental nodes have been deployed.

Let the initial network topology be represented as an undirected simple graph $G = (V, E)$, where V is the set of nodes, and there exists an edge $(u, v) \in E$ if and only if u and v can directly communicate. Denote the potential network by the undirected simple graph $G' = (V', E') = (V \cup P, E \cup E_P)$. The set of possible locations where supplemental

Symbol	Description
A_{uv}	binary indicator variable that potential edge $(u, v) \in E_P$ is in the actual network so $(u, v) \in E_D$
b_s^i	source requirement of $s \in S_i$
$D \subseteq P$	deployment of supplemental nodes
D_p	binary indicator variable that $p \in D$
E	set of edges
E_P	set of potential edges, i.e., edges with at least one endpoint in P
$G = (V, E)$	initial network with nodes V and edges E
$G' = (V \cup P, E \cup E_P)$	potential network with initial nodes V, potential nodes P, initial edges E, and potential edges E_P
$G_D = (V_D, E_D)$	actual network for a deployment D: the subgraph of G' induced by $V \cup D$
h	maximum number of supplemental nodes allowed in MaxSG-NA
K	number of source groups
m	number of edges
n	number of nodes
$S \subset V$	set of source nodes
$S_i \in \mathfrak{S}$	ith group of source nodes
$\mathfrak{S} = \{S_1, \ldots, S_K\}$	set of groups of agents that partitions the agents
$T \subset V \cup P \setminus S$	set of sink nodes
V	set of initial nodes
$x_i \in T$	sink node assigned to group i, if any
X_{it}	binary indicator variable that sink $t \in T$ is assigned to group $S_i \in \mathfrak{S}$

Table 3.1: Summary of notation for Chapter 3

nodes be deployed is denoted by P, with $V\ P = \emptyset$, and the set of potential communication links is denoted by E_P, with $E_P\ E = \emptyset$. For each edge $(u, v) \in E_P$, either $u \in P$ or $v \in P$ or both. If $u \in P$ and $v \notin P$, then $(u, v) \in E_P$ represents that a supplemental node positioned at u can communicate with node v; similarly, if $u \notin P$ and $v \in P$, then $(u, v) \in E_P$ means that a supplemental node positioned at v can communicate with node u; and finally, if both $u, v \in P$, then if supplemental nodes were deployed to both u and v, they would be able to communicate.

We represent a deployment of supplemental nodes by $D \subseteq P$, the set of potential locations where we have deployed supplemental nodes. We denote the actual network instantiated by D by the undirected simple graph $G_D = (V_D, E_D)$ that is the subgraph of G' induced by $V \cup D$.

Let S be the set of source nodes, and $\mathfrak{S} = \{S1, S2, \ldots, S_K\}$ be the set of source groups, where $S_i \subseteq S$ for $i = 1, \ldots, K$. We would like all of the nodes within a group $S_i \in S$ to be connected to each other in G_D. We refer to this property as group connectivity, and say that the group S_i is connected (or is a connected group).

There are two related network augmentation optimization problems, depending on the objective function. The Minimum Deployment Connected Groups problem seeks to minimize the number of supplemental nodes required to satisfy all groups. The Maximum Connected Groups problem seeks to maximize the number of satisfied groups given a fixed number of supplemental nodes.

Problem 4. Minimum Deployment Connected Groups (MinDepCG). Given an initial network $G = (V, E)$, potential network $G' = (V \cup P, E \cup E_P)$, and source groups $\mathfrak{S} = \{S_1, \ldots, S_K\}$, find a deployment D with minimum $|D|$ such that all of the groups are connected in G_D.

Problem 5. Maximum Connected Groups (MaxCG). Given an initial network $G = (V, E)$, potential network $G' = (V \cup P, E \cup E_P)$, source groups $\mathfrak{S} = \{S_1, \ldots, S_K\}$, and integer constant $h \geq 1$, find a deployment D with $|D| \leq h$ such that the maximum number of groups are connected in G_D.

The decision problem related to MinDepCG and MaxCG is the Connected Groups problem:

Problem 6. Connected Groups (CG). Given an initial network $G = (V, E)$, potential network $G' = (V \cup P, E \cup E_P)$, agent groups $\mathfrak{S} = \{S_1, \ldots, S_K\}$, and integer constants $h \geq 1$ and $k \geq 1$, is there a deployment D with $|D| \leq h$ such that at least k groups are connected in G_D?

We next show that the Connected Groups problem is NP-complete, even for a single group.

Theorem 4. *CG is NP-complete.*

Proof. To show this we must show that CG is in NP and is NP-hard. It is clearly in NP, as we can easily verify in polynomial time whether a set of possible locations $D \subseteq P$ is a solution or not.

Proof of NP-hardness is done by reduction from the graph Steiner tree (ST) decision problem with unit-weight edges, which is known to be NP-complete [20]. The input to ST is an undirected graph $G_{ST} = (V_{ST}, E_{ST})$, a set of terminals $X_{ST} \subseteq V_{ST}$, and a maximum cost k_{ST}. The problem is to decide whether there exists a tree $T_{ST} = (W_{ST}, F_{ST})$ which is a subgraph of G_{ST} with $X_{ST} \subseteq W_{ST}$ and at most k_{ST} edges. Such a tree T_{ST} is called a *Steiner tree*.

From an ST input instance G_{ST}, X_{ST}, k_{ST} for ST, we construct the following CG input $G, G', \mathfrak{S}, h$:

- $G = (V, E)$ where $V = X_{ST}$, and $E = \{(u, v) \in E_{ST} | u \in X_{ST} \text{ and } v \in X_{ST}\}$.
- $G' = (V \cup P, E \cup E_P)$, where $P = V_{ST} \setminus X_{ST}$ and $E_P = \{(u, v) \in E_{ST} | u \notin X_{ST} \text{ or } v \notin X_{ST}\}$.
- $\mathfrak{S} = \{S_1\}$, where $S_1 = X_{ST}$.
- $h = k_{ST} + 1 - |X_{ST}|$.
- $k = 1$.

We now show that G_{ST} has a Steiner tree with at most k_{ST} edges if and only if the constructed CG instance has a solution.

Suppose first that G_{ST} has a Steiner tree $T_{ST} = (W_{ST}, F_{ST})$ with at most k_{ST} edges. We will show that there is a deployment that satisfies the solution conditions for the CG instance. Set $D = W_{ST} \setminus X_{ST}$. Then clearly

$$\begin{aligned} D &= W_{ST} \setminus X_{ST} \\ &\subseteq V_{ST} \setminus X_{ST} && (X_{ST} \subseteq V_{ST}) \\ &= P && \text{(by construction)} \end{aligned}$$

Note that $S_1 = X_{ST}$ is connected in the subgraph of G' induced by $V \cup D$, since X_{ST} is connected in T_{ST}.

We also get that

$$\begin{aligned} |W_{ST}| &= |F_{ST}| + 1 && (T_{ST} \text{ is a tree}) \\ &\leq k_{ST} + 1 && \text{(by assumption)} \end{aligned}$$

so that

$$\begin{aligned}
|D| &= |W_{ST} \setminus X_{ST}| \\
&= |W_{ST}| - |X_{ST}| && (X_{ST} \subseteq W_{ST}) \\
&\leq k_{ST} + 1 - X_{ST} \\
&= h && \text{(by construction)}
\end{aligned}$$

Thus D is a solution to CG.

Assume now instead that there is a solution D to a CG instance as constructed above. We will show that there must then be a Steiner tree with at most k_{ST} edges for the Steiner instance. Let $G_D = (V_D, E_D)$ denote the subgraph of G' induced by D. Then

$$\begin{aligned}
|V_D| &= |V_D \cap X_{ST}| + |V_D \setminus X_{ST}| \\
&= |X_{ST}| + |V_D \setminus X_{ST}| && (X_{ST} \subseteq V_D) \\
&= |X_{ST}| + |V_D \cap P| && (P = V_{ST} \setminus X_{ST}) \\
&= |X_{ST}| + |D| \\
&\leq |X_{ST}| + h && (D \text{ is a solution to CG}) \\
&= |X_{ST}| + k_{ST} + 1 - |X_{ST}| && \text{(by construction)} \\
&= k_{ST} + 1.
\end{aligned}$$

Because D is a solution to CG, the nodes in $S_1 = X_{ST}$ are connected in G_D. Let $T_D = (V_D, E_{T_D})$ denote a spanning tree of G_D. Because the nodes in X_{ST} are connected in G_D, it follows that they are also connected in T_D, and so T_D is a Steiner tree. Now we get that

$$\begin{aligned}
|E_{T_D} &= |V_D| - 1 && (T_D \text{ is a tree}) \\
&\leq k_{ST} + 1 - 1 && \text{(by above)} \\
&= k_{ST}
\end{aligned}$$

Thus T_D is a Steiner tree with at most k_{ST} edges.

Hence G_ST has a Steiner tree with at most k_{ST} edges if and only if the constructed CG instance has a solution. Therefore CG is NP-complete. □

Because other variations of the Network Augmentation problem with the maximum group or minimum deployment objective functions are generalizations of this simple case, it follows that they too are NP-complete.

3.2 Network Augmentation and Flow Allocation

We now consider the full problem of network augmentation and flow allocation. This is the problem of choosing a deployment and selecting a sink and satisfying flow for each group. It is a generalization of both the pure network augmentation and flow allocation problems.

As in pure network augmentation, we are given an initial network $G = (V, E)$ and potential network $G' = (V', E') = (V \cup P, E \cup E_P)$, but these are now capacitated, directed graphs, with the capacity of each edge $e \in E'$ denoted by w_e, as in the flow allocation problem. Capacities may be either positive, real values (representing limited capacity), or may be positive infinity (representing unlimited capacity). A deployment $D \subseteq P$ is the set of potential location where supplemental nodes are deployed. The actual network for a deployment D is the graph $G_D = (V_D, E_D)$ that is the subgraph of G' induced by $V_D = V \cup D$.

As in pure flow allocation, source nodes $S \subset V$ are partitioned into groups $\mathfrak{S} = \{S_1, \ldots, S_K\}$, each of which must send flow to a single sink. The flow requirement b_s^i is the amount of flow that must be sent by source node $s \in S_i$. The set of sinks is denoted by $T \subset V \cup P$. If $v \in T \cap V$, it means that the node v can act as a sink, while if $p \in T \cap P$, it means that a supplemental node deployed to p can act as a sink. The set of groups for which flow has been allocated is $\mathcal{X} \subseteq \mathfrak{S}$, and the sink assigned to group $S_i \in \mathcal{X}$ is $x_i \in T$. The amount of flow on edge (u, v) for group S_i is denoted f_{uv}^i.

A feasible solution is a deployment D, sink assignments x, and flow f such that f is a satisfying flow for $\mathcal{X}$ in the actual network G_D. This means that none of the groups can be assigned to sinks that are not in G_D, and no flow can be transmitted on any potential edge that is not in G_D. Formally, these constraints are

$$x_i \in T \cap (V \cup D) \qquad \forall S_i \in \mathcal{X} \tag{3.1}$$

$$f_{uv}^i = 0 \qquad \forall (u, v) \notin E_D, \forall S_i \in \mathcal{X}. \tag{3.2}$$

These constraints must be met in addition to the usual constraints in Equations (2.1) – (2.3) for satisfying flows.

As in pure network augmentation, there are two optimization problems depending on the choice of objective function. Maximum Satisfied Groups with Network Augmentation (MaxSG-NA) is the problem of satisfying the greatest number of groups with a fixed number of supplemental nodes. The Minimum Deployment (MinDep) problem is to satisfy all of the groups using the fewest number of supplemental nodes possible.

Problem 7. Maximum Satisfied Groups with Network Augmentation (MaxSG-NA).

Given initial network $G = (V, E)$, potential network $G' = (V', E') = (V \cup P, E \cup E_P)$, capacities w, source groups $\mathfrak{S} = \{S_1, \ldots, S_K\}$, source requirements b, sinks $T \subset V \cup P$, and number of supplemental nodes $h > 0$, find a deployment D, subset of groups $\mathcal{X} \subseteq \mathfrak{S}$, sink assignment x, and satisfying flow f for $\mathcal{X}$ in G_D such that $|\mathcal{X}|$ is maximized

Problem 8. Minimum Deployment (MinDep). Given initial network $G = (V, E)$, potential network $G' = (V', E') = (V \cup P, E \cup E_P)$, capacities w, source groups $\mathfrak{S} = \{S_1, \ldots, S_K\}$, source requirements b, and sinks $T \subset V \cup P$, find a deployment D, sink assignment x, and flow f for $\mathcal{X}$ in G_D and satisfying flow f for $\mathfrak{S}$ in G_D such that $|D|$ is minimized.

Theorem 5. *MaxSG-NA and MinDep are NP-hard.*

Proof. This follows from MaxSG-NA and MinDep generalizing MaxSG and CG, both of which are NP-hard. □

3.3 Solving MaxSG-NA and MinDep

In this section we present algorithms for solving MaxSG-NA and MinDep. To solve the problems optimally we use a MILP formulation.

3.3.1 Optimal Algorithms

We optimally solve MaxSG-NA and MinDep by formulating the problems as MILPs which can then be solved using standard MILP solvers. This approach simultaneously optimizes both the network augmentation and flow allocation parts of the problems. The central issue is to convert the constraints in Equations (3.1) and (3.2) into linear constraints. To represent the deployment D, we use binary indicator variables D_p for $p \in P$, where $D_p = 1$ if $p \in P$ and $D_p = 0$ otherwise. As in MILP 1 for flow allocation, we represent sink assignments using binary indicator variables X_{it} for all $S_i \in \mathfrak{S}$, $t \in T$, where $X_{it} = 1$ if t is the sink for group S_i and $X_{it} = 0$ otherwise.

In representing the constraint in Equation (3.1), note that the definition of the X_{it} variables already constrains x_i to be in T, and furthermore if $t \in V$ then there is no need for an additional constraint. For $t \in P$, we would like to express the constraint that X_{it} can take the value 1 only if $D_t = 1$. This can be expressed by the constraint $X_{it} \leq D_t$ for all $t \in T \cap P$. We must also ensure that at most one sink is assigned to each group; this is expressed as $\sum_{t \in T} X_{it} \leq 1$ for all $i \in [1..K]$, just as in MILP 1. There is no need to

explicitly represent $\mathcal{X}$, because that the composition of $\mathcal{X}$ is already determined by the X_{it} variables: $\mathcal{X}$ contains group S_i if and only if $X_{it} = 1$ for some $t \in T \cap V_D$.

The constraint in Equation (3.2) requires that flow only be sent on edges in the actual network G_D. There is no need to modify the flow capacity constraint for edges in the initial network, as they are always included in the actual network. For each potential edge $e = (u, v) \in E_P$, we add a binary indicator variable A_{uv} (also denoted A_e) that will be 1 if (u, v) is present in G_D, and 0 otherwise. We then constrain the total amount of flow on (u, v) to be no more than $w_{uv}A_{uv}$, which is equal to w_{uv} if $(u, v) \in E_D$ and 0 otherwise, as desired. This expression is linear as w_{uv} is an input parameter.

We can distinguish two kinds of potential edges: those that have a potential location at only one end point, and those that have potential locations at both end points. An edge with a potential location at only one end point has the form $e = (u, p) \in E_P$ where $u \in V$ and $p \in P$, or $e = (p, v) \in E_P$ where $v \in V$ and $p \in P$. In either of these cases, the presence of the e in G_D is indicated by the presence of p in G_D, and so we constrain A_e to equal D_p.

An edge with potential locations at both end points is slightly trickier to represent. In this case the edge has the form $(p, q) \in E_P$ where both p and q are in P. Thus the expression for the presence or absence of e in the actual network is D_pD_q, which is 1 if they are both present and 0 otherwise. Unfortunately, D_pD_q is not linear but quadratic, and so it cannot be used in a MILP constraint. Instead we introduce a binary indicator variable A_{pq} for $(p, q) \in E_P$ with both $p, q \in P$. We constrain $A_{pq} \leq D_p$ and $A_{pq} \leq D_q$; this ensures that A_{pq} must be 0 unless both $p, q \in D$, in which case it can be 1. We further add a constraint $A_{pq} \geq D_p + D_q - 1$; this ensures that when $p, q \in D$ then A_{pq} must be 1, and it can be 0 otherwise. Together these guarantee that A_{pq} is 1 when both $p, q \in D$, and 0 otherwise. Now we can constrain the total flow on (p, q) to be at most $w_{pq}A_{pq}$.

MILP 2 solves MaxSG-NA by extending MILP 1 for flow allocation to include the network augmentation constraints of Equations (3.1) and (3.2) as described above. The objective in Equation (3.3) is to maximize the number of assigned groups, while Equation (3.15) limits the number of supplemental nodes that can be used to at h. The constraint in Equation (3.1) is captured by Equation (3.13), while the constraint in Equation (3.2) is captured by Equations (3.7) – (3.12).

MILP 3 modifies MILP 2 to solve the MinDep problem instead. The objective in Equation (3.20) is to minimize the number of supplemental nodes deployed. The MaxSG-NA constraint in Equation (3.15) limiting the deployment size is removed from MILP 3, and the MaxSG-NA constraint in Equation (3.14) limiting at most one sink to be assigned to each group is changed to the constraint that each group must have exactly one sink as-

MILP 2 MILP formulation of MaxSG-NA.

$$\underset{f,X,D,A}{\text{Maximize}} \sum_{i=1}^{k} \sum_{t \in T} X_{it} \tag{3.3}$$

subject to:

$$\sum_{(u,v) \in E} f_{uv}^{i} - \sum_{(v,u) \in E} f_{vu}^{i} = b_u^i \sum_{t \in T} X_{it} \qquad \forall u \in V, i \in [1..K] \tag{3.4}$$

$$\sum_{(u,v) \in E} f_{uv}^{i} - \sum_{(v,u) \in E} f_{vu}^{i} = X_{it} \sum_{v \in V} b_v^i \qquad \forall t \in T, i \in [1..K] \tag{3.5}$$

$$\sum_{i=1}^{K} f_e^i \leq w_e \qquad \forall e \in E \tag{3.6}$$

$$\sum_{i=1}^{K} f_e^i \leq w_e A_e \qquad \forall e \in E_P \tag{3.7}$$

$$A_{pv} = D_p \qquad \forall (p,v) \in E_P, p \in P, v \notin P \tag{3.8}$$

$$A_{up} = D_p \qquad \forall (u,p) \in E_P, p \in P, u \notin P \tag{3.9}$$

$$A_{pq} \leq D_p \qquad \forall (p,q) \in E_P, p \in P, q \in P \tag{3.10}$$

$$A_{pq} \leq D_q \qquad \forall (p,q) \in E_P, p \in P, q \in P \tag{3.11}$$

$$A_{pq} \geq D_p + D_q - 1 \qquad \forall (p,q) \in E_P, p \in P, q \in P \tag{3.12}$$

$$X_{ip} \leq D_p \qquad \forall p \in P, i \in [1..K] \tag{3.13}$$

$$\sum_{t \in T} X_{it} \leq 1 \qquad \forall i \in [1..K] \tag{3.14}$$

$$\sum_{p \in P} D_p \leq h \tag{3.15}$$

$$f_e^i \geq 0 \qquad \forall e \in E, i \in [1..K] \tag{3.16}$$

$$A_{uv} \in \{0,1\} \qquad \forall (u,v) \in E_P \tag{3.17}$$

$$D_p \in \{0,1\} \qquad \forall p \in P \tag{3.18}$$

$$X_{it} \in \{0,1\} \qquad \forall i \in [1..K], t \in T \tag{3.19}$$

signed to it as shown in Equation (3.31), so that all groups are satisfied in the MinDep problem.

The number of variables and constraints in both MILP 2 and MILP 3 is polynomial in the size of the potential network G'. However, because mixed integer programming is NP-hard, there is no known polynomial-time algorithm for solving these formulations and running times are likely to be exponential in general.

3.3.2 Heuristic Algorithms

To help cope with the exponential running times of solving MaxSG-NA and MinDep optimally, we develop heuristic algorithms that iterate through the groups and deploy supplemental nodes as necessary to satisfy the groups.

Most of the work is done by the subroutine SolveGroup, shown in Algorithm 2. This extends a partial solution by finding a minimum deployment needed to satisfy a group S_j given the partial solution for previously considered groups $F \setminus \{S_j\}$. It does this by solving a modified version of the MinDep MILP but with the following changes:

1. Variables are only solved for the groups in F, with constraints updated accordingly.
2. A new constraint to extend the previous deployment: $D'_p = 1$ for all $p \in D$ (Equation (3.49)).
3. A new constraint to extend the previous sink assignment: $X_{it} = 1$ for all $S_i \in F$ where $x_i = t$ (Equation (3.48)).

The basic iterative heuristic for MinDep is shown in Algorithm 3. The while loop in lines 4 – 7 iterates through the source groups. In each iteration, one of the groups that has not yet been processed is selected (line 5) and the current solution extended by calling SolveGroup (line 7).

We consider three variations on this heuristic that depends on the choice of the next group to process in line 5:

1. MD-Rand-H. The next group is chosen at random.
2. MD-MF-H. The group that requires largest increase in the deployment size is chosen next. The intuition behind this "most first" heuristic is that this group will only become harder to satisfy on future iterations and so it should be satisfied as soon as possible.
3. MD-LF-H. The group that requires the smallest increase in the deployment size is chosen next. The intuition behind this "least first" heuristic is that greedily choos-

MILP 3 MILP formulation of MinDep.

$$\underset{f,X,D,A}{\text{Minimize}} \sum_{p \in P} D_p \tag{3.20}$$

subject to:

$$\sum_{(u,v) \in E} f^i_{uv} - \sum_{(v,u) \in E} f^i_{vu} = b^i_u \sum_{t \in T} X_{it} \qquad \forall u \in V, i \in [1..K] \tag{3.21}$$

$$\sum_{(u,v) \in E} f^i_{uv} - \sum_{(v,u) \in E} f^i_{vu} = X_{it} \sum_{v \in V} b^i_v \qquad \forall t \in T, i \in [1..K] \tag{3.22}$$

$$\sum_{i=1}^{K} f^i_e \leq w_e \qquad \forall e \in E \tag{3.23}$$

$$\sum_{i=1}^{K} f^i_e \leq w_e A_e \qquad \forall e \in E_P \tag{3.24}$$

$$A_{pv} = D_p \qquad \forall (p,v) \in E_P, p \in P, v \notin P \tag{3.25}$$

$$A_{up} = D_p \qquad \forall (u,p) \in E_P, p \in P, u \notin P \tag{3.26}$$

$$A_{pq} \leq D_p \qquad \forall (p,q) \in E_P, p \in P, q \in P \tag{3.27}$$

$$A_{pq} \leq D_q \qquad \forall (p,q) \in E_P, p \in P, q \in P \tag{3.28}$$

$$A_{pq} \geq D_p + D_q - 1 \qquad \forall (p,q) \in E_P, p \in P, q \in P \tag{3.29}$$

$$X_{ip} \leq D_p \qquad \forall p \in P, i \in [1..K] \tag{3.30}$$

$$\sum_{t \in T} X_{it} = 1 \qquad \forall i \in [1..K] \tag{3.31}$$

$$f^i_e \geq 0 \qquad \forall e \in E, i \in [1..K] \tag{3.32}$$

$$A_{uv} \in \{0,1\} \qquad \forall (u,v) \in E_P \tag{3.33}$$

$$D_p \in \{0,1\} \qquad \forall p \in P \tag{3.34}$$

$$X_{it} \in \{0,1\} \qquad \forall i \in [1..K], t \in T \tag{3.35}$$

Algorithm 2 SolveGroup

Input: MaxSG Instance $\mathcal{I}$, current group index j, set of processed groups F, current deployment D, current sink assignments x

Output: Extended deployment D', extended sink assignment x', flow f

1: Solve MILP:

$$\underset{f,X,D',A}{\text{Minimize}} \sum_{p \in P} D'_p \tag{3.36}$$

subject to:

$$\sum_{(u,v) \in E} f^i_{uv} - \sum_{(v,u) \in E} f^i_{vu} = b^i_u \sum_{t \in T} X_{it} \qquad \forall u \in V, S_i \in F \tag{3.37}$$

$$\sum_{(u,v) \in E} f^i_{uv} - \sum_{(v,u) \in E} f^i_{vu} = X_{it} \sum_{v \in V} b^i_v \qquad \forall t \in T, S_i \in F \tag{3.38}$$

$$\sum_{S_i \in F} f^i_e \leq w_e \qquad \forall e \in E \tag{3.39}$$

$$\sum_{S_i \in F} f^i_e \leq w_e A_e \qquad \forall e \in E_P \tag{3.40}$$

$$A_{pv} = D'_p \qquad \forall (p,v) \in E_P, p \in P, v \notin P \tag{3.41}$$

$$A_{up} = D'_p \qquad \forall (u,p) \in E_P, p \in P, u \notin P \tag{3.42}$$

$$A_{pq} \leq D'_p \qquad \forall (p,q) \in E_P, p \in P, q \in P \tag{3.43}$$

$$A_{pq} \leq D'_q \qquad \forall (p,q) \in E_P, p \in P, q \in P \tag{3.44}$$

$$A_{pq} \geq D'_p + D'_q - 1 \qquad \forall (p,q) \in E_P, p \in P, q \in P \tag{3.45}$$

$$X_{ip} \leq D'_p \qquad \forall p \in P, S_i \in F \tag{3.46}$$

$$\sum_{t \in T} X_{it} = 1 \qquad \forall S_i \in F \tag{3.47}$$

$$X_{it} = 1 \qquad \forall S_i \in F \text{ where } x_i = t \tag{3.48}$$

$$D'_p = 1 \qquad \forall p \in D \tag{3.49}$$

$$f^i_e \geq 0 \qquad \forall e \in E, i \in [1..K] \tag{3.50}$$

$$A_{uv} \in \{0,1\} \qquad \forall (u,v) \in E_P \tag{3.51}$$

$$D'_p \in \{0,1\} \qquad \forall p \in P \tag{3.52}$$

$$X_{it} \in \{0,1\} \qquad \forall i \in [1..K], t \in T \tag{3.53}$$

2: Set $D' \leftarrow \{p \in P : D'_p = 1\}$

3: Set x' with $x' \leftarrow x$ and $x'_j \leftarrow t$ for $X_{jt} = 1$

4: **return** (D', x', f)

Algorithm 3 MD-Iter-H: Basic iterative heuristic for MinDep.

Input: Instance $\mathcal{I} = (G, G', w, \mathfrak{S}, b, h)$: initial network $G = (V, E)$, potential network $G' = (V \cup P, E \cup E_P)$, capacities w, sinks T, groups $\mathfrak{S} = \{S_1, \ldots, S_K\}$, source requirements b.
Output: Solution $\mathcal{O} = (D, x, f)$: deployment D, sink assignments x, and flows f.

```
D ← ∅ (set of deployed positions)
Initialize x as empty assignment
F ← ∅ (set of processed groups)
while F ≠ S do
    Choose S_j ∈ 𝔖 \ F
    F ← F ∪ {S_j}
    (D, x, f) ← SolveGroup(ℐ, j, F, D, x)
return (D, x, f)
```

ing the smallest increase in deployment size is a good way to minimize the total deployment size.

Algorithm 4 is the basic iterative heuristic for MaxSG-NA. It is identical to the heuristic for MinDep, except that on each iteration it only extends the solution if this does not exceed the limit on the number of supplemental nodes (lines 8 – 9). As with MD-Iter-H, we consider three variations of SG-Iter-H depending on the choice in line 5: SG-Rand-H, SG-MF-H, and SG-LF-H.

3.4 Experiments

In this section we present results of empirical evaluation of our algorithms. We randomly generated input instances and ran the algorithms on them. Each data point reported here is the average over 50 input instances.

Input instances were generated based on a disk graph model. In the results presented here, 100 initial nodes and 64 potential locations were uniformly distributed over a square region of the plane measuring 250×250. All initial nodes had a range of 25, and all supplemental nodes had a range of 50. The initial network $G = (V, E)$ was formed by adding an edge between two initial nodes if the distance between them was at most the range (25). The bandwidth of all edges in E was 10. The potential network $G' = (V \cup P, E \cup E_P)$ was formed by adding an edge between a potential location and another node (initial node or another potential location) whenever the distance between them was

Algorithm 4 SG-Iter-H: Basic iterative heuristic for MaxSG-NA.

Input: Instance $\mathcal{I} = (G, G', w, \mathfrak{S}, b, h)$: initial network $G = (V, E)$, potential network $G' = (V \cup P, E \cup E_P)$, capacities w, sinks T, groups $\mathfrak{S} = \{S_1, \ldots, S_K\}$, source requirements b, number of supplemental nodes h.

Output: Solution $\mathcal{O} = (D, x, f)$: deployment D, sink assignments x, and flows f.

$D \leftarrow \emptyset$ (set of deployed positions)
Initialize x as empty assignment
$F \leftarrow \emptyset$ (set of processed groups)
while $F \neq S$ **do**
  Choose $S_j \in \mathfrak{S} \setminus F$
  $F \leftarrow F \cup \{S_j\}$
  $(D', x', f') \leftarrow$ SOLVEGROUP($\mathcal{I}, j, F, D, x$)
  if $|D'| < h$ **then**
    $D \leftarrow D', x \leftarrow x', f \leftarrow f'$
return (D, x, f)

at most the supplemental node range (50). This reflects cases where supplemental nodes have longer ranges due to factors such as higher-gain antennas or environmental factors less path loss due to reflection and obstructions between airborne supplemental nodes. The capacity of all edges in E_P was 50.

Group sizes were independently and uniformly distributed between 2 and 7, inclusive, and each input instance could have groups of different sizes. Group membership was chosen independently and uniformly at random from all possible groups of the appropriate size.

The source requirements of each group member was a real number independently and uniformly distributed between 10 and 30. Deployments up to a maximum size of 4 supplemental nodes were considered in these experiments.

To solve the MILPs, we used CPLEX 10.0, a commercial solver, running on a computer with a 3 GHz Pentium IV processor and 1 GB of RAM.

We solved MinDep problems both optimally using the exact MILP formulation and heuristically using MD-Rand-H, MD-MF-H, and MD-LF-H. All algorithms were run for a maximum of 2 hours on each instance. The average deployment sizes found by the optimal and heuristic algorithms for 1 to 6 groups is plotted in Figure 3.1. The three heuristics found deployments of similar sizes, although the greedy heuristics significantly outperformed the random heuristic for larger numbers of groups. The optimal algorithm found significantly smaller deployments than the heuristics, with the difference in deployment sizes increasing with the number of groups. However, even for 6 groups, MD-Rand-H

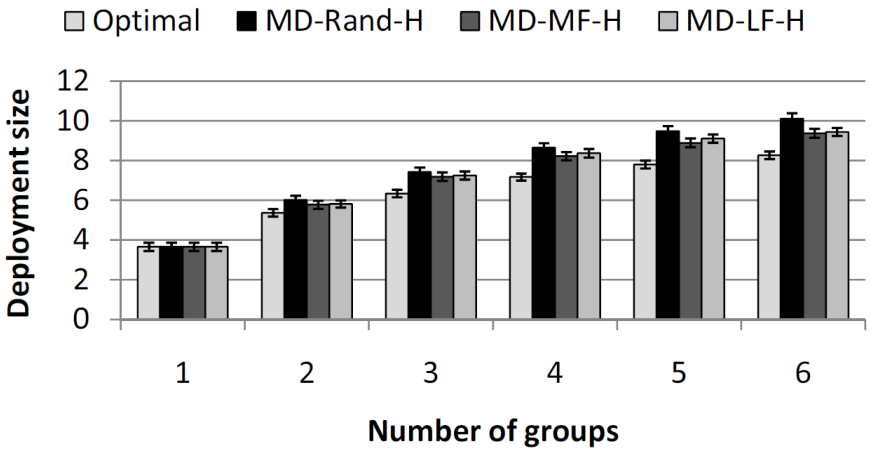


Figure 3.1: Average deployment sizes found by Algorithm 3

found solutions within 25% of optimal on average, while the MD-MF-H and MD-LF-H found solutions within 14% of optimal on average.

The improvement in MinDep solution quality of the optimal algorithm over the heuristics comes at the cost of increased running time. The average running times are shown in Figure 3.2, along with the standard errors. While the running times of the optimal algorithm and the heuristics are very close for 1 or 2 groups, they quickly diverge for larger number of groups as the running time of the optimal algorithm increases dramatically. The error bars show that the variability in running times of the optimal algorithm also increases with the number of groups, while the iterative heuristic exhibits less variability that increases more slowly than the optimal algorithm. This suggests that heuristics may be better suited in cases where the highly variable running times is undesirable.

The running time of the optimal algorithm increases with the number of groups, and this rate increases from 1 to 5 groups, then seems to decrease from 5 to 6 groups. However, this is an artifact resulting from the maximum cut-off time of 2 hours, which introduces an artificial cap on the running time of MILP 3. As the number of groups increases, the proportion of instances that can be solved optimally within 2 hours decreases from 100% for 1 and 2 groups down to about 50% for 6 groups, as shown in Figure 3.3. The heuristics all terminated within 2 hours and so are not plotted on that figure.

We also compared the SG-LF-H to the optimal solution computed by MILP 2 for 1 to 5 groups. Figure 3.4 shows the number of groups satisfied by deployments of up to 4 supplemental nodes found by SG-LF-H, normalized to the optimal number of groups satisfied

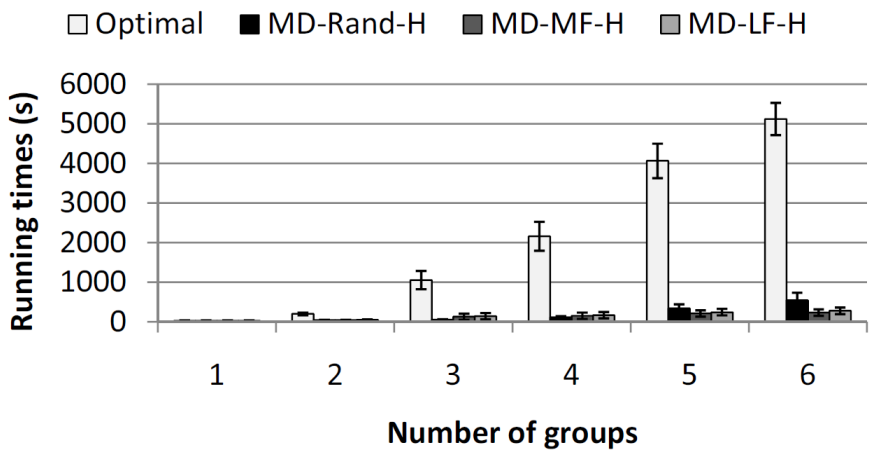


Figure 3.2: Running times for solving MinDep using MILP 3 and Algorithm 3.

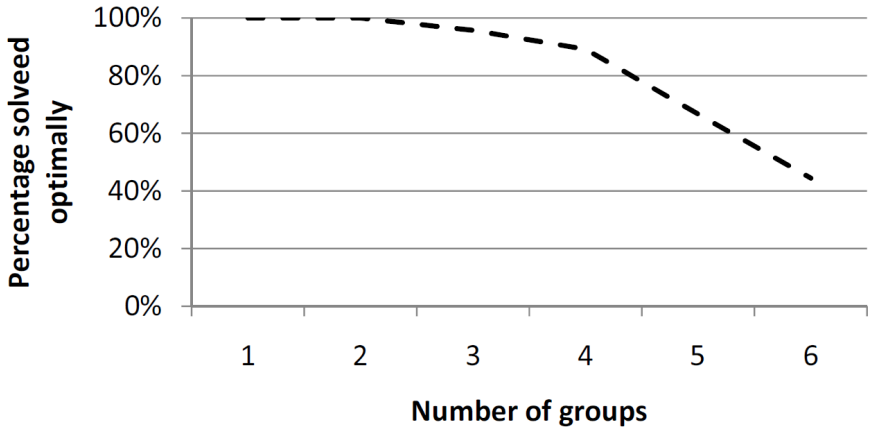


Figure 3.3: Proportion of MinDep instances solved optimally by MILP 3 within two hours.

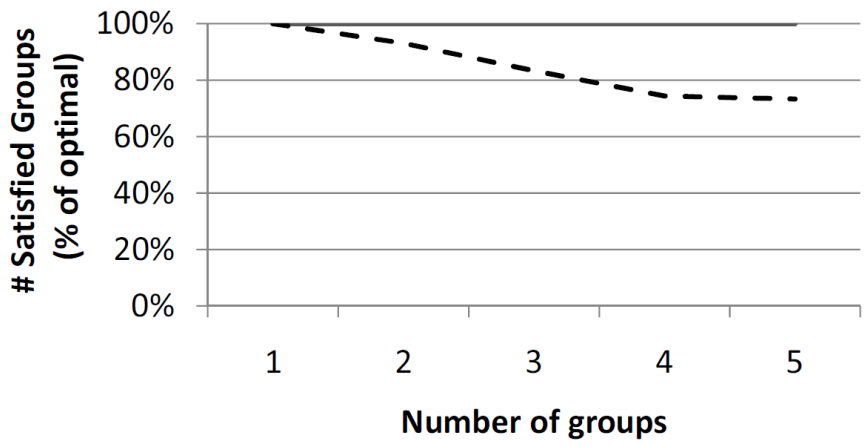


Figure 3.4: Average number of groups satisfied by Algorithm 4 as a percentage of the optimal found by MILP 2.

as determined by MILP 2. As can be seen, SG-LF-H initially performs well, but solution quality decreases relative to the optimal as the number of groups increases. A comparison of the running times of the optimal MILP and SG-LF-H is given in Figure 3.5. Both algorithms require similar amounts of time for problems with 1 and 2 groups, but the running time of the optimal algorithm for increases sharply thereafter, while the running time of SG-LF-H increases much more slowly.

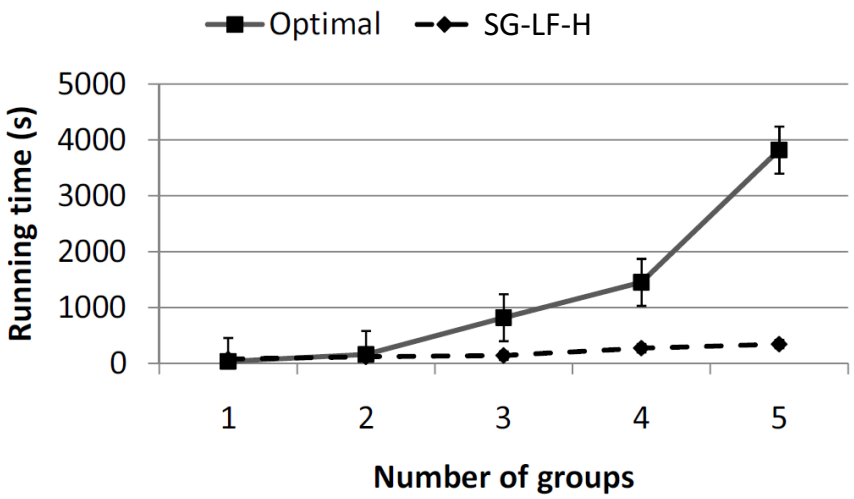


Figure 3.5: Running times for solving MaxSG-NA using MILP 2 and SG-LF-H.

Chapter 4

Flow Allocation in Adversarial Environments: Zero-Sum Games

In the previous chapter we saw how the agents could augment the environment in order to improve the quality of flow allocation for task execution by improving connectivity and reducing capacity bottlenecks. In this chapter we shift from the capacitated environments examined in the previous chapters and consider costly environments, where there is no limit on the amount of flow that can be transmitted on each edge, but flows instead incur costs proportional to their amounts. We shift our attention also from environments that can be changed by the agents to improve performance, to one that can be changed by a hostile adversary to decrease the performance of the agents by increasing their costs.

We start by considering the extreme case where the interests of the agents and the adversary are diametrically opposed. In this case the adversary is truly malicious because its only interest is to impose greater costs on the agents. For example, in an urban robot setting robots can avoid locations that might be observed by cameras placed by the adversary, but doing so causes them to take more circuitous routes and thus suffer higher travel costs. The robots balance the tradeoff between not being observed with the higher travel costs. A truly malicious adversary does not merely care about observing the robots but also the travel costs of the robots, and is only interested in forcing the robots' tradeoff to be as bad as possible, even if the robots' best course of action ends up avoiding observation by the adversary completely. This kind of strictly competitive payoffs arises very naturally in the sensor network domain where the adversary jams nodes in order to force them to use more precious battery power in stronger transmissions or a greater number of retransmissions. The sensor nodes want to minimize total battery depletion while the adversary seeks to maximize it, and it doesn't matter if this battery depletion occurs because nodes transmit

through the jamming or around it.

This strictly competitive setting is naturally modeled as a two-player zero-sum game played on a graph. We assume that the agents work together cooperatively, and therefore represent them by a single player, called the *sender*. The other player is the adversary. In this chapter we formally describe two zero-sum games.

In the *path game*, the sender's pure strategies are combinations of paths (unsplittable flows) from multiple known starting locations to a common, known destination. In the multirobot domain, these represent paths taken by the robots through the environment. The adversary chooses a subset of attacks to play from a set of possible attacks. In the multirobot domain where the adversary has k cameras, each attack corresponds to the placement of a single camera, while a pure strategy is a particular deployment of k cameras to locations in the environment to observe the robots' movements. The payoffs for the players are quantified by the *harm* suffered by the sender and is computed by a *harm function* mapping strategy profiles to harm. Although zero-sum games can be solved in polynomial time, this is polynomial in size of the pure strategy spaces. In the path game, the strategy space of the sender can be exponential in the size of the graph, while the strategy space of the adversary may be exponential in the number of simultaneous attacks. This makes a direct application of the traditional linear programming techniques for zero-sum games impractical for the path game.

The *network flow game* addresses this difficulty. This game differs from the path game in the sender's strategy space. Rather than choosing discrete paths, the sender chooses divisible network flows from source nodes to a common sink node. This naturally represents domains where the quantity moving through the graph can be divided. For example, in wireless sensor networks traffic can be divided over multiple paths. We further show in Section 4.3 that the network flow game is equivalent to the path game for a class of harm functions that can be represented using *harm matrices*. Intuitively, the attacks for these harm functions correspond to a set of costs on the edges of the graph, with the harm for multiple attacks additively combined. In such cases the network flows represent marginal probabilities of edges being included in mixed sender strategies of the path game. By using a compact representation based on the marginal probabilities of sender and adversary strategies, rather than full probability distributions over the pure strategies of the path game, we are able to find equilibria in polynomial time. We describe a simple technique for sampling a pure strategy for the path game from the network flow representation. In the next chapter we show how this approach can be leveraged to solve non-zero sum games that include both attack costs and movement costs.

A summary of the notation used in this chapter is provided in Table 4.1. Portions of the work in this chapter was originally published in Okamoto, et al. [47].

Symbol	Description
A	set of possible attacks for the adversary
$\mathcal{A}$	adversary strategy space, $\{A' \subseteq A : \lvert A' \rvert \leq k\}$
b_s	source requirement of $s \in S$
$\Delta(\cdot)$	the categorical probability distribution over its argument, a finite set
E	set of edges
f	sender strategy, a flow with f_{uv} being the amount of flow on edge $(u, v) \in E$
$\mathcal{F}$	zero-sum network flow game sender strategy space, the set of feasible flows from S to t
$G = (V, E)$	network with nodes V and edges E
$\mathcal{H}_{\text{PG}}$	harm function for the zero-sum path game
k	number of attacks that the adversary can play simultaneously
m	number of edges
M	harm matrix with M_{ij} the harm from sending 1 unit of flow on $e_j \in E$ when the adversary plays $a_i \in A$
n	number of nodes
$\mathcal{P}$	zero-sum path game sender strategy space, $\mathcal{P}_1 \times \mathcal{P}_2 \times \ldots \mathcal{P}_{\lvert S \rvert}$
$\mathcal{P}_i$	set of all paths from $s_i \in S$ to t
q	an adversary mixed strategy, often expressed as a vector of marginal probabilities over A
S	set of source nodes
t	the sink
V	set of nodes
$\zeta \in \mathcal{A}$	adversary pure strategy, a set of attacks

Table 4.1: Summary of notation for Chapter 4

4.1 Zero-Sum Path Game (ZS-PG)

We first consider the normal form of the zero-sum game. The zero-sum path game (ZS-PG) is played between a sender and an adversary taking actions on a directed[1] graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges. A subset of the nodes $S \subset V$ are the *source nodes* while another of the nodes $t \in V \setminus S$ is the *sink*. Without loss of generality, we assume that there are no incoming edges to any source node. (If there is such a source node s, add a new source node s' to S and an outgoing edge (s', s) to E, and remove s from S.) Each source node has a weight $b_s > 0$ representing the value of that source node. The sender chooses a path π_s from each source $s \in S$ to the sink t (we assume that such a path exists for every $s \in S$), and his pure strategy space is set of all combinations of paths from S to t, $\mathcal{P}$.

The adversary has a finite set of attacks A, and can carry out up to k attacks from A simultaneously. Thus the adversary's set of pure strategies $\mathcal{A}$ is the set of all subsets of A of size at most k, which has size $\Theta(|A|^k)$. We assume that both players have full knowledge of G, S, t, A, and k.

The payoff in this game is quantified by the *harm* suffered by the sender as a result of one or more of his paths being attacked. As a zero-sum game, we assume that the sender seeks to minimize the harm suffered, while the adversary seeks to maximize the harm inflicted. In general, harm may be an arbitrary function $\mathcal{H}_{\text{PG}} : \mathcal{P} \times \mathcal{A} \to [0, +\infty)$ mapping from the sender's choice of paths and the adversary's choice of attacks to a non-negative number. For convenience, we overload the notation to extend the harm function to mixed strategies, with

$$\mathcal{H}_{\text{PG}}(p, q) = \sum_{\pi \in \mathcal{P}} \sum_{\zeta \in \mathcal{A}} p_\pi q_\zeta \mathcal{H}_{\text{PG}}(\pi, \zeta)$$

representing the *expected harm* when the sender plays mixed strategy $p \in \Delta(\mathcal{P})$ and the adversary plays mixed strategy $q \in \Delta(\mathcal{A})$.

We can now give the normal form of the zero-sum path game.

Definition 1. The zero-sum path game is a game $(2, (\mathcal{P}, \mathcal{A}), (-\mathcal{H}_{\text{PG}}, \mathcal{H}_{\text{PG}}))$ where

- $\mathcal{P} = \mathcal{P}_1 \times \mathcal{P}_2 \times \cdots \times \mathcal{P}_{|S|}$ with $\mathcal{P}_i$ the set of all paths from $s_i \in S$ to t.
- $\mathcal{A} = \{A' : A' \subseteq A \wedge |A'| \leq k\}$.
- $\mathcal{H}_{\text{PG}}$ is the harm function.

[1]The model and algorithms can be applied to undirected graphs in a straightforward manner.

We are interested in finding a mixed strategy Nash equilibrium strategy profile for the sender and adversary. Because this is a zero-sum game, this corresponds to the minimax and maximin strategies. The sender's minimax problem is shown below, where $p \in \Delta(\mathcal{P})$ and $q \in \Delta(\mathcal{A})$.

$$\min_{p} \max_{q} \mathcal{H}_{\text{PG}}(p, q) \tag{4.1}$$

subject to

$$\sum_{\pi \in \mathcal{P}} p_\pi = 1 \tag{4.2}$$

$$\sum_{\zeta \in \mathcal{A}} q_\zeta = 1 \tag{4.3}$$

$$p_\pi \geq 0 \qquad \forall \pi \in \mathcal{P} \tag{4.4}$$

$$q_\zeta \geq 0 \qquad \forall \zeta \in \mathcal{A} \tag{4.5}$$

We can use the well-known linear programming techniques for finding such strategy profiles for zero-sum games in time polynomial in the size of $\mathcal{P}$ and $\mathcal{A}$, but these are problematic because $\mathcal{P}$ is exponential in n and $|S|$, and $|\mathcal{A}| = \binom{|A|}{k} = \Theta(n^k)$ in the worst case.

Instead, we exploit the structure of a broad class of harm functions to decompose the payoff function into a polynomially-sized representation, called a harm matrix. This reduces the computational time of finding equilibria, while still being able to represent the payoffs for a wide variety of harm functions that have not been considered using other representations. Although the algorithm we describe here is specific to the zero-sum game of strictly competitive payoffs, in Chapter 5 we will leverage the network flow approach to solving certain kinds of non-zero-sum games for different types of harm matrices that we present in the next section.

4.2 Harm matrices

Harm matrices are applicable when the harm function can be decomposed so that harm can be computed independently for each pair of edge and attack, then summed to calculate the total harm. The harm matrix M is a matrix with n rows and m columns, where M_{ij} is the amount of harm suffered by the sender if the adversary plays attack $a_i \in A$ and the sender chooses a path with edge e_j. Intuitively, each attack specifies non-negative weights on the edges, the length of an edge is the sum of the weights from the attacks, and harm is

the sum of the lengths of the paths. The following provides the precise conditions required of the harm function decomposition.

Let $\pi \in \mathcal{P}$ be a pure strategy of the sender with π^s denoting the path from source node s to t, and let $\zeta \in \mathcal{A}$ be a pure strategy of the adversary. Many interesting and realistic harm functions can be decomposed in the following way. First, the total harm is the sum of harm for the individual paths from each source to the sink:

$$\mathcal{H}_{\text{PG}}(\pi, \zeta) = \sum_{s \in S} h(\pi_s, \zeta) \tag{4.6}$$

Second, the harm for a set of attacks is the sum of harm for the individual attacks in ζ:

$$h(\pi_s, \zeta) = \sum_{a \in \zeta} h(\pi_s, a) \tag{4.7}$$

Finally, the harm for π^s is the sum of the harm for the individual edges in π_s, and this harm is a linear function of the weight of s, with the specific linear function depending on the edge e and attack a through a constant value α_{ae}:

$$h(\pi_s, a) = \sum_{e \in \pi_s} \alpha_{ae} b_s. \tag{4.8}$$

For harm functions that satisfy these properties, we construct a harm matrix M with $|A|$ rows and m edges where entry $M_{ij} = \alpha_{a_i e_j}$.

We now turn our attention to several interesting harm functions and their harm matrix representation. For the purposes of exposition we consider a radio jamming attack, in which the adversary transmits radio signals to disrupt the sender's communications. This attack increases latency because packets need to be retransmitted. If multiple points on a pathway are disrupted by the attack, additional latency is suffered, resulting in additive harm.

Consider the simple *path intersection* harm function, where uniform harm is suffered if and only if a node on the pathway is attacked, as arises if the adversary must directly jam a node on a pathway to disrupt communication, and each attack causes the same increase in latency. In this case the attacks correspond to nodes in V and the harm is represented by the following harm matrix:

$$M_{ij}^{\text{path int}} = \begin{cases} 1 & \text{if } e_j = (u, v_i) \text{ for some node } u \in V \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

More generally, different nodes may be more or less susceptible to jamming, leading to different amounts of harm being suffered when the sender routes through an attacked node. This is represented by a harm matrix with heterogeneous values:

$$M_{ij}^{\text{gen path int}} = \begin{cases} c_i & \text{if } e_j = (u, v_i) \text{ for some node } u \in V \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

Jamming is not the only cause of latency. Latency is also incurred on every transmission on an edge on the path chosen by the sender. This is an example of *costly transmission* in which the sender incurs a cost every time he utilizes an edge, irrespective of which nodes the adversary has attacked; examples include latency or battery power usage. By using a harm matrix that includes both the harm suffered from the adversary's attacks and the transmission costs, the sender can rationally reason about the tradeoff between them. Such a harm matrix may use homogeneous cost values (biasing toward pathways with fewer hops) or heterogeneous cost values that depend on the edge, representing characteristics such as requiring more battery power to transmit to more distant nodes or to nodes in high noise areas. In the following harm matrix, harm c_j (cost of transmission) is incurred for every unit of flow transmitted on edge e_j.

$$M_{ij}^{\text{cost trans}} = \begin{cases} 1 + c_j & \text{if } e_j = (u, v_i) \text{ for some node } u \in V \\ c_j & \text{otherwise} \end{cases} \tag{4.11}$$

There is also no requirement that paths must intersect the attacked nodes for harm to be suffered. In the jamming example, attacking a node in a wireless network may also jam neighboring nodes, even if those nodes are not communicating with the attacked node. Thus transmissions on all edges to neighbors of an attacked node incur increased latency.

$$M_{ij}^{\text{nonlocal}} = \begin{cases} c_j & \text{if } e_j = (u, v) \text{ and } (v, v_i) \in E \\ 0 & \text{otherwise} \end{cases} \tag{4.12}$$

In then next subsection we describe the network flow game that explicitly uses the harm matrix structure along with marginal probability distributions to efficiently represent the strategy spaces of the sender and adversary.

4.3 Zero-Sum Network Flow Game (ZS-NFG)

In the zero-sum network flow game, the sender chooses flows from the source nodes to the sink, sending b_s units of flow from each source $s \in S$ to the sink t. The flows are represented by an $m \times 1$ column vector f, where f_{uv} is the amount of flow sent on edge (u, v); when denoting an edge simply as $e \in E$, we also use the notation f_e. The sender's strategy space $\mathcal{F}$ is the set of all feasible flows from the source nodes to the sink, that is, all f such that

$$\sum_{(v,u)\in E} f_{vu} = b_v + \sum_{(u,v)\in E} f_{uv} \qquad \forall v \in V \setminus \{t\} \tag{4.13}$$

$$f_{uv} \geq 0 \qquad \forall (u, v) \in E. \tag{4.14}$$

Equation 4.13 is the flow conservation constraint requiring outgoing flow for all nodes (other than the sink) to be equal to the source requirement of the node plus the sum of the incoming flow. Equation (4.14) ensures that the flow on each edge is non-negative[2]. Flows may be divided on alternate paths from the source nodes to the sink, leading to a continuous strategy space for the sender if there are at least two paths from any source node to the sink. The adversary's strategies in ZS-NFG are exactly the same as those in ZS-PG, with his pure strategy set $\mathcal{A}$ being all subsets of A of up to k attacks, for a total of $\Theta(|A|^k)$ pure strategies.

The payoffs in ZS-NFG are quantified by the harm as in ZS-PG, but in ZS-NFG we only consider harm functions that can be represented using harm matrices. For a harm matrix M, sender pure strategy $f \in \mathcal{F}$, and adversary pure strategy $\zeta \in \mathcal{A}$ the harm is

$$H(f, \zeta) = \sum_{a\in\zeta} \text{row}_a[M] f \tag{4.15}$$

where $\text{row}_a[M]$ is the $1 \times m$ row vector corresponding to the row of M for attack $a \in A$. For convenience, we also overload this notation to finite mixed strategies of the sender and adversary, with the expected harm

$$H(p, q) = \sum_{f\in F} \sum_{\zeta\in\mathcal{A}} p_f q_\zeta H(f, \zeta),$$

[2]Other network flow formulations sometimes represent the *net flow* between nodes, which requires that $f_{uv} = -f_{vu}$. Our formulation instead uses the actual, non-negative amount of flow sent on each directed edge.

where $p \in \Delta(F)$ for finite $F \subset \mathcal{F}$ and $q \in \Delta(\mathcal{A})$.

We are interested in finding Nash equilibrium, which, because of the zero-sum payoffs, are strategy profiles where the sender plays a strategy that minimizes the maximum harm (i.e., the sender plays a minimax strategy) and the adversary plays a strategy that maximizes the minimum harm (i.e., the adversary plays a maximin strategy). We consider equilibria where the sender is a playing a pure flow strategy (i.e., some $f \in \mathcal{F}$) while allowing the adversary to play a mixed strategy $q \in \Delta(\mathcal{A})$. Before continuing, we must prove that such an equilibrium must exist.

We first prove that for any mixed sender strategy over a finite set of flows there exists a pure sender strategy with the same expected payoff when played against any adversary pure strategy.

Lemma 1. *Given a finite set of flows $F \subset \mathcal{F}$, a mixed sender strategy $p \in \Delta(F)$, and an adversary pure strategy $\zeta \in \mathcal{A}$, there exists a flow $f \in \mathcal{F}$ such that*

$$H(f, \zeta) = \sum_{f' \in F} p_{f'} H(f', \zeta).$$

Proof. We prove this by construction. Set the amount of flow f_e on each edge e to be the expected amount of flow under the mixed strategy p. This is easily shown using vector addition:

$$f \triangleq \sum_{f' \in F} p_{f'} f' \tag{4.16}$$

We can now verify that the payoff with f is equal to the expected payoff with p.

$$\begin{aligned} H(f, \zeta) &= \sum_{a \in \zeta} \text{row}_a[M] f \\ &= \sum_{a \in \zeta} \text{row}_a[M] \sum_{f' \in F} p_{f'} f' \\ &= \sum_{f' \in F} p_{f'} \sum_{a \in \zeta} \text{row}_a[M] f' \\ &= \sum_{f' \in F} p_{f'} H(f', \zeta). \end{aligned}$$

□

As a corollary, the lemma is applicable to mixed adversary strategies, due to the linearity

of expectation and the harm function.

Corollary. *Given a finite set of flows $F \subset \mathcal{F}$, a mixed sender strategy $p \in \Delta(F)$, and an adversary mixed strategy $q \in \Delta(\mathcal{A})$, there exists a flow $f \in \mathcal{F}$ such that*

$$H(f, q) = \sum_{f' \in F} p_{f'} H(f', q).$$

Theorem 1. *Consider a zero-sum path game with a harm function that can be represented by a harm matrix, and a zero-sum network flow game with the same harm matrix. Then the following must hold:*

1. *For all sender mixed strategies $p \in \Delta(\mathcal{P})$ of the ZS-PG there exists a sender pure strategy $f \in \mathcal{F}$ of the ZS-NFG such that for all adversary mixed strategies $q \in \Delta(\mathcal{A})$, the expected harm in the two games is the same.*
2. *For all sender pure strategies $f \in \mathcal{F}$ of the ZS-NFG there exists a sender mixed strategy $p \in \Delta(\mathcal{P})$ of the ZS-PG such that for all adversary mixed strategies $q \in \Delta(\mathcal{A})$, the expected harm in the two games is the same.*

Proof. We prove the first claim by construction. Let $p \in \Delta(\mathcal{P})$ and $q \in \Delta(\mathcal{A})$. Recall the definition of $\mathcal{P}$ from Definition 1: $\mathcal{P} = \mathcal{P}_1 \times \cdots \times \mathcal{P}_{|S|}$ where $\mathcal{P}_i$ is the set of all paths from $s_i \in S$ to t. Without loss of generality we restrict the $\mathcal{P}_i$ to include only simple paths. For each path $\pi_{s_i} \in \mathcal{P}_i$ we construct a corresponding flow, f^{π_i} that sends the full amount of flow b_{s_i} from s_i to t along the edges of π_i:

$$f_e^{\pi_i} \triangleq \begin{cases} b_s & \text{if } e \in \pi_i \\ 0 & \text{otherwise} \end{cases} \qquad \forall e \in E. \tag{4.17}$$

For each $\pi = (\pi_1, \ldots, \pi_{|S|}) \in \mathcal{P}$, we then construct a flow f^π that sends the full amount of flow b_s along π_i for each $s_i \in S$. This is achieved by summing the flows from each source:

$$f_e^\pi \triangleq \sum_{s_i \in S} f_e^{\pi_i} \qquad \forall e \in E \tag{4.18}$$

We can now define a set F of flows in the ZS-NFG that correspond to the ZS-PG pure strategies:

$$F \triangleq \{f^\pi : \pi \in \mathcal{P}\}$$

Because $\mathcal{P}$ is finite, F must be finite as well. We extend the ZS-PG mixed strategy p to a

ZS-NFG mixed strategy $p' \in \Delta(F)$ by setting $p'_{f^\pi} \triangleq p_\pi$ for all $\pi \in \mathcal{P}$. We then show that the expected harm of (p, q) in the ZS-PG is the same as the expected harm of (p', q) in the ZS-NFG. The expected harm in the ZS-PG is

$$\begin{aligned}
\mathcal{H}_{\text{PG}}(p, q) &= \sum_{\pi \in \mathcal{P}} \sum_{\zeta \in \mathcal{A}} p_\pi q_\zeta \sum_{s \in S} \sum_{a_i \in \zeta} \sum_{e_j \in \pi_s} M_{ij} b_s \\
&= \sum_{\pi \in \mathcal{P}} \sum_{\zeta \in \mathcal{A}} p_\pi q_\zeta \sum_{a_i \in \zeta} \sum_{s \in S} \sum_{e_j \in \pi_s} M_{ij} b_s \\
&= \sum_{\pi \in \mathcal{P}} \sum_{\zeta \in \mathcal{A}} p_\pi q_\zeta \sum_{a_i \in \zeta} \sum_{e_j \in E} M_{ij} f^\pi_{e_j} \\
&= \sum_{\pi \in \mathcal{P}} \sum_{\zeta \in \mathcal{A}} p_\pi q_\zeta \sum_{a_i \in \zeta} \text{row}_{a_i}[M] f^\pi \\
&= \sum_{\pi \in \mathcal{P}} \sum_{\zeta \in \mathcal{A}} p_\pi q_\zeta H(f^\pi, \zeta) \\
&= \sum_{f^\pi \in F} \sum_{\zeta \in \mathcal{A}} p'_{f^\pi} q_\zeta H(f^\pi, \zeta) \\
&= H(p', q),
\end{aligned}$$

which is the expected harm in the ZS-NFG.

By Lemma 1, there exists a pure ZS-NFG sender strategy $f \in \mathcal{F}$ with the same expected harm as p', and thus with the same expected harm as p in the ZS-PG. Thus the first claim is proved.

We prove the second claim by construction as well. Let $f \in \mathcal{F}$ and $q \in \Delta(\mathcal{A})$. For each $\pi_{s_i} \in \mathcal{P}_i$ we set

$$\Pr(\pi_{s_i}) \triangleq \prod_{(u,v) \in \pi_{s_i}} \frac{f_{uv}}{b_u + \sum_{(u',u) \in E} f_{u'u}} \tag{4.19}$$

and for $\pi \in \mathcal{P}$ we set

$$p_\pi \triangleq \prod_{s_i \in S} \Pr(\pi_{s_i}). \tag{4.20}$$

□

Theorem 1 establishes a very close relationship between ZS-PG and ZS-NFG: in essence, the two games model the same strategic interaction. In other words, ZS-NFG can be seen

as an alternate representation for the normal form representation used by ZS-PG. As we will see, this representation can be solved more efficiently than the normal form of ZS-PG. We explore this connection further in Section 4.5, but for the moment we note an important corollary of Theorem 1, the existence of an equilibrium for ZS-NFG.

Corollary. *The zero-sum network flow game has a Nash equilibrium* (f, q) *where the sender plays a pure strategy* $f \in \mathcal{F}$ *and the adversary plays a mixed strategy* $q \in \Delta(\mathcal{A})$.

Proof. This follows from Theorem 1 and the existence of a Nash equilibrium for ZS-PG as a finite strategy game. □

Note that the corollary guarantees the existence of an equilibrium where the sender plays a pure flow strategy, allowing us to restrict our search to pure strategies for the sender. Although there may be multiple equilibria, the game is zero-sum and so we know that there are no equilibrium selection issues: the payoffs (i.e., the harm) in all equilibria are the same, and the equilibria are all interchangeable. Moreover, we know that the equilibria must satisfy the minimax conditions: the sender's flow must minimize the maximum harm over all possible pure strategies played by the adversary, and the adversary's mixed strategy must maximize the minimum harm over all possible pure strategies played by the sender. We can write the sender's network flow minimax problem as

$$\underset{f \in \mathcal{F}}{\text{Minimize}} \ \max_{\zeta \in \mathcal{A}} \sum_{a \in \zeta} \text{row}_a[M] f \tag{4.21}$$

subject to

$$\sum_{u:(v,u)\in E} f_{vu} = b_v + \sum_{u:(u,v)\in E} f_{uv} \qquad \forall v \in V \setminus \{t\} \tag{4.22}$$

$$f_{uv} \geq 0 \qquad \forall (u,v) \in E. \tag{4.23}$$

Equation (4.22) is the flow conservation constraint requiring outgoing flow for all nodes (other than the sink) to be equal to the source requirement of the node plus the sum of the incoming flow. Equation (4.23) requires all flows to be non-negative.

It is helpful to gain some intuition into the structure of the network flow game equilibria. Assume that we are using the homogeneous path intersection harm matrix (so that attacks correspond to nodes). From the sender's perspective, his choice f is a best response to the adversary if he can't improve it by changing some of the flow from one of the current paths to a better path, i.e., a path with lower probability of intersecting a node under attack. Note that if the sender is sending flow on a path π_s from $s \in S$ to t, and there is a path with lower probability π'_s from s to t, then he should move *all* of the flow from π_s to π'_s. Let π_s be a path from one of the $s \in S$ to t, and let $f^*_{\pi_s}$ denote the best response amount of flow

sent on π_s. We can write the sender's best response property:

$$f^*_{\pi_s} > 0 \implies \sum_{v \in \pi} q_v = \min_{\pi'_s \text{ from } s \text{ to } t} \sum_{v \in \pi'_s} q_v \tag{4.24}$$

Thus, the adversary should evenly distribute probability among nodes so that all paths from one or more source nodes to the sink have equal probability of being attacked.

Now for the adversary, a distribution q is a best response if he can't improve it by choosing a different distribution q' with greater harm. Fixing the sender's strategy f, the adversary's payoff for q is qMf and so a rational adversary will choose to put all of her probability on the nodes with the corresponding greatest harm. Let $(Mf)_v$ denote the element for node v in column vector Mf, then

$$p^*_v > 0 \implies (Mf)_v = \max_{v' \in V} (Mf)_{v'} \tag{4.25}$$

Hence, the sender should minimize the maximum $(Mf)_v$ in order to minimize the maximum harm that will be caused by the adversary. We call the term $(Mf)_v$ the *potential harm* because it represents the amount of harm that could be suffered if the adversary attacked node v.

As a consequence of these two properties, the sender will route flows to distribute the potential harm $(Mf)_v$ as evenly as possible. When we are considering the path intersection harm matrix $M^{\text{path int}}$, the sender's equilibrium strategy effectively performs network load balancing. Furthermore, the set of nodes with maximum $(Mf)_v$ form a vertex cut in the network separating a subset of the source nodes from the sink. While graph theoretic approaches (e.g., [66]) can find such vertex cuts for the case of a single source and single sink and the simple path intersection harm function, they are incapable of handling more complex problems. Our approach is unique in balancing the potential harm rather than just the network flow.

Because this vertex cut U will contain the nodes with maximum $(Mf)_v$, the adversary will only attack nodes in the vertex cut, and by virtue of it being a vertex cut, the adversary can attack those nodes to guarantee that all paths from a subset of source nodes $S' \subseteq S$ on the other side of the cut must go through an attacked node. Given U, there is a minimum subset U' which is still a vertex cut. U' has the further property that it contains no redundant nodes in the sense that no node can be removed and still guarantee that all paths from S' to t pass through it. Thus the adversary will assign non-zero probability to the nodes in U', evenly divided so that all paths have equal probability of being attacked.

4.4 Computing Network Flow Equilibrium

In this section we describe the linear program for finding the equilibrium strategies. When the adversary can execute a single attack, the sender's equilibrium strategy can be found via a straightforward linearization of the sender's network flow minimax problem, LP 1. The harm is represented by the variable H and is constrained to be at least as great as the

LP 1 Network Flow LP for $k = 1$.

$$\underset{f,H}{\text{Minimize}}\ H \tag{4.26}$$

subject to:

$$H \geq \text{row}_a[M]f \qquad \forall a \in A \tag{4.27}$$

$$\sum_{(v,u)\in E} f_{vu} = b_v + \sum_{(u,v)\in E} f_{uv} \qquad \forall v \in V \setminus \{t\} \tag{4.28}$$

$$f_{uv} \geq 0 \qquad \forall (u,v) \in E \tag{4.29}$$

harm resulting from any single attack by Eq. (4.27). The term $\text{row}_a[M]$ denotes the row corresponding to attack a in matrix M. LP1 has $|E| + 1$ variables and $2n - 1$ constraints. Thus it can be solved in polynomial time (with respect to n) [8].

We now extend LP 1 to allow the adversary to execute multiple attacks simultaneously. This program introduces an additional $|A|$ variables, the λ_a. We first observe that when

LP 2 Network Flow LP for general $k \geq 1$

$$\underset{f,H,\lambda}{\text{Minimize}}\ kH + \sum_{a\in A} \lambda_a \tag{4.30}$$

subject to:

$$\sum_{(v,u)\in E} f_{vu} = b_v + \sum_{(u,v)\in E} f_{uv} \qquad \forall v \in V \setminus \{t\} \tag{4.31}$$

$$H \geq \text{row}_a[M]f - \lambda_a \qquad \forall a \in A \tag{4.32}$$

$$f_{uv} \geq 0 \qquad \forall (u,v) \in E \tag{4.33}$$

$$\lambda_a \geq 0 \qquad \forall a \in A \tag{4.34}$$

$k = 1$, LP 2 reduces back to LP 1, as expected, with all $\lambda_a = 0$. When $k > 1$, the potential harm for each attack $a \in A$ is $(Mf)_a = \text{row}_a[M]f$. As a best response the adversary will attack a set $Z \subset A$ of size k that causes maximum harm by choosing the k attacks with greatest potential harm. If we let H denote the kth largest potential harm, it follows that the total harm will be

$$\begin{aligned}\sum_{a \in Z} \text{row}_a[M]f &= \sum_{a \in Z} (\text{row}_a[M]f - H) + H \\ &= kH + \sum_{a \in Z} (\text{row}_a[M]f - H).\end{aligned}$$

LP 2 minimizes this total harm over all possible such Z by letting λ_a be the variable for $\text{row}_a[M]f - H$.

The adversary's equilibrium strategy can be found by solving LP 3, the dual program of LP 2. There are $|A|$ of the q variables, with q_a for $a \in A$ being the marginal probability

LP 3 Adversary Network Flow LP for general $k \geq 1$

$$\text{Maximize}_{r,q} \sum_{s \in S} b_s r_s \tag{4.35}$$

subject to:

$$r_u - r_v \leq q^T \text{col}_{(u,v)}[M] \qquad \forall (u,v) \in E, u \neq t, v \neq t \tag{4.36}$$

$$-r_v \leq q^T \text{col}_{(t,v)}[M] \qquad \forall (t,v) \in E \tag{4.37}$$

$$r_u \leq q^T \text{col}_{(u,t)}[M] \qquad \forall (u,t) \in E \tag{4.38}$$

$$\sum_{v \in V} q_v = k \tag{4.39}$$

$$0 \leq q_v \leq 1 \qquad \forall v \in V \tag{4.40}$$

of the adversary playing a strategy that includes attack a. By Eq. (4.39), the sum of these marginal probabilities must equal k because the adversary can execute k attacks simultaneously. There is a variable r_v for each $v \in V \setminus \{t\}$, representing the least amount of harm that the sender could suffer sending one unit of flow starting from v. This is constrained by Eq. (4.36) which says that for an edge $(u, v) \in E$, the least harm the sender can suffer starting from u can be no more than the harm suffered crossing the edge (u, v) added to the least harm starting from v. The notation $\text{col}_{(u,v)}[M]$ denotes the column for edge (u, v) in matrix M. The objective is to maximize the harm starting from the source nodes, weighted

by the amount of flow that must be sent from each source node.

4.5 Using ZS-NFG to Solve ZS-PG

The network flow game represents cases where the sender is moving divisible flow through the network. This is applicable for cases where the assets moved through the network really are divisible, as may be the case with communication traffic or large quantities of physical goods in transportation convoys. Despite the infinite strategy space for the sender, LP 2 provides a way to find an equilibrium strategy in polynomial time with respect to G and A.

In addition to modeling cases of divisible flow, the zero-sum network flow game models the strategic interactions in the zero-sum path game with a harm matrix, as shown in Theorem 1: every mixed strategy $p \in \Delta(\mathcal{P})$ in the ZS-PG has a corresponding pure strategy $f \in \mathcal{F}$ in the ZS-NFG that has exactly the same payoff behavior against the adversary, and vice versa. Thus we can use LP 2 to find an equilibrium sender strategy for ZS-NFG, and we know that there is a corresponding equilibrium sender strategy p for ZS-PG.

To see why the ZS-NFG flow-based representation is so much more efficient than the ZS-PG normal form representation, we turn to the properties of harm functions that can be represented by harm matrices, and to the proof of Theorem 1. By Equation (4.6), the harm for each source node can be computed independently, with the total harm being the sum over all source nodes. Likewise, in the proof of the first claim of Theorem 1 for each ZS-PG pure strategy $\pi \in \mathcal{P}$ we compute a flow f^{π_i} for each source s_i using Equation (4.17), then sum them to get the total flow f^{π} as shown in Equation (4.18). The amount of flow f_e^{π} on each edge e is is the number of paths that include e, weighted by the weights b of the source nodes.

For a ZS-PG mixed strategy $p \in \mathcal{P}$ the flow is the sum of the flows for each pure strategy, weighted by the probability of that pure strategy, as constructed by Equation (4.16). This means that the amount of flow f_e on each edge e is the expected value of the number of paths that include e, weighted by the weights of the source nodes. When $b = 1$ this is just the expected number of paths that include e, and by linearity the component f_e^s due to each source s is the marginal probability that e is on a path played from s to t. This is the key to ZS-NFG providing an exponentially more efficient representation of the sender's strategy than ZS-PG: because of the harm matrix structure of the harm function, it suffices to represent only the marginal probabilities of edges being used, instead of the full joint probability distribution over all possible combinations of paths.

To find a mixed strategy $p \in \Delta(\mathcal{P})$ of ZS-PG that corresponds to a flow f, we can explicitly represent the probability of each $\pi \in \mathcal{P}$ with a variable p_π and write a system of linear equations that relates the joint probability distribution p to the marginal probability distribution f. In general there may be many possible mixed strategies of ZS-PG that correspond to each flow f, which will result in this system of equations having multiple possible solutions, but all of these solutions will have the same payoff behavior because they are all equilibria. However, it is obvious that solving for p in this manner is impractical as the number of variables is $|\mathcal{P}|$, which may be exponential in the size of G in general. In fact, just specifying p completely may require exponential time in general.

Thus, instead of explicitly solving for p given f, we instead *sample* a pure strategy $\pi \in \mathcal{P}$ from the distribution p, without ever computing p directly. The construction of p in the proof of the second claim of Theorem 1 is one specific mixed strategy of ZS-PG that corresponds to f, and it is easy to use for sampling. According to Equation (4.20), we can sample paths from each source node independently. This is convenient for applications, as agents represented by the source nodes do not need to coordinate with each other once they are provided with the flow f. The probability distribution of paths for each source node is given by Equation (4.19) and is also very convenient. Starting from the source node, the next edge to take is chosen probabilistically according to the fraction of the total outgoing flow of the current node on each edge. Thus the agent starts at the source node and executes a Markovian policy where the state is the current node, and the transition to the next state is determined probabilistically by the fraction of outgoing flow to each neighbor. This provides a very natural, polynomial time way to operationalize the equilibrium computed by LP 2

An adversary pure strategy can also be sampled from the marginal probabilities computed in LP3. This involves sampling sets of k attacks from A and can be accomplished in polynomial time by using weighted random sampling [17] with the marginal probabilities as weights, or comb sampling [62].

4.6 Experiments

4.6.1 Simulation setup

We simulated a multiagent system in which agents (nodes) were distributed uniformly at random in a 50×50 region of the plane. The agents could communicate using a multi-hop network, where agents within a Euclidean distance of 10 were neighbors in the network. This resulted in a network with many paths between any two nodes on average, but where

the graph is not fully connected (in which case the problem is trivial). In this section, the number of agents is denoted by n, the number of source nodes by s, and the number of nodes that can be attacked by the adversary by k. Each source node had a flow distributed uniformly at random between 5 and 20. Each point in the figures is an average over 100 randomly generated instances.

Results

We performed three sets of experiments using the CPLEX 10.0.1 solver on a Linux machine with a 2.40 GHz Intel Core 2 processor with 4GB of RAM. The first set of experiments present the average harm found by LP 2 as we increase the network size and the number of source nodes. Figure 4.1 shows the network size on the x-axis and the harm on the y-axis. The 5 lines in the graph correspond to the number of source nodes s increasing from 1 to 5. k was set to 1 in this experiment. From the graph we obtain that as the number of nodes in the network increase and other parameters stay constant, the harm decreases. This is expected because the sender can spread the flow across many more nodes thus decreasing the harm caused by the adversary at any particular node. The same trend is observed across all values of s.

For network sizes less than 500 nodes, the harm increases with the number of source nodes because more source nodes imply more flow. For network sizes greater than 500, there is a very small difference in the harm for varying source nodes because the sender can spread the flow across a large number of paths, thus decreasing the ability of the adversary to cause harm at any single node.

Figure 4.2 studies the effect on running times of LP 2 as the network size and number of source nodes increase. The number of nodes in the network is shown on the x-axis and the average running times in seconds is plotted on the y-axis. The 5 lines in the figures correspond to the various source node settings. The plot shows that as the number of source nodes increase the running time increases, but even at 1000 nodes the running time is on the order of a second. This shows that LP 2 is a fast algorithm for finding the equilibrium in large networks.

Our next experiment studies the solution quality and runtime results for LP 2 for $k = 1$ and for k between 10 to 50 in increments of 10, as shown in Table 4.2. Note that $n = 600$ and $s = 3$ for this experiment. From the table we obtain that as k increases the harm caused in the network increases as expected. In fact, for each jump of 10 nodes in k, the harm caused also increases fairly uniformly i.e. about 5 units. However, the running time is unaffected by an increasing k.

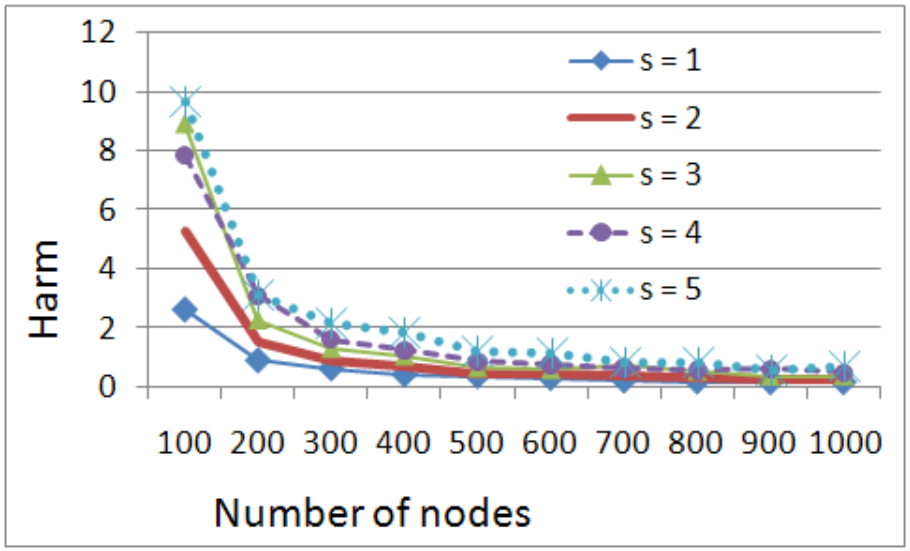


Figure 4.1: Harm as a function of network size and number of source nodes

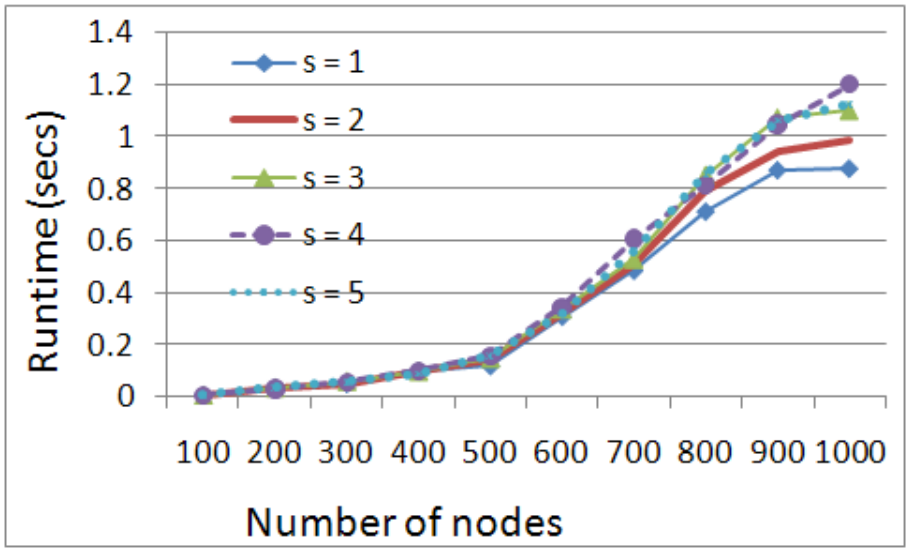


Figure 4.2: Running times for increasing network size and source nodes

k	Harm	Runtime
1	.583	.344
10	5.825	.343
20	11.651	.337
30	16.960	.339
40	21.950	.338
50	26.212	.373

Table 4.2: Effect of varying k on harm and runtime.

Our last experiment highlights ability of LP 2 to balance harm and network performance. We evaluated LP 2 with the costly communication harm matrix, and measured the average number of hops in the resulting pathways. We compared to the RANGER algorithm [62], an algorithm that maximizes security but does not take network performance into account, and a shortest paths (SP) algorithm that optimizes for network performance without regard for security. The results are shown in Figure 4.3, for RANGER, SP, and varying values of the parameter c with larger values indicating more costly communication. RANGER finds paths two orders of magnitude greater than LP 2 because it optimizes only for spreading the flow as evenly as possible. Because it is insensitive to the network performance, it will perform arbitrarily bad as c increases. In contrast, LP 2 tends to find short paths when communication is costly, and when communication is very costly ($c = 1$ means a single hop is as harmful as an adversary's attack), LP 2 converges to the shortest paths strategy.

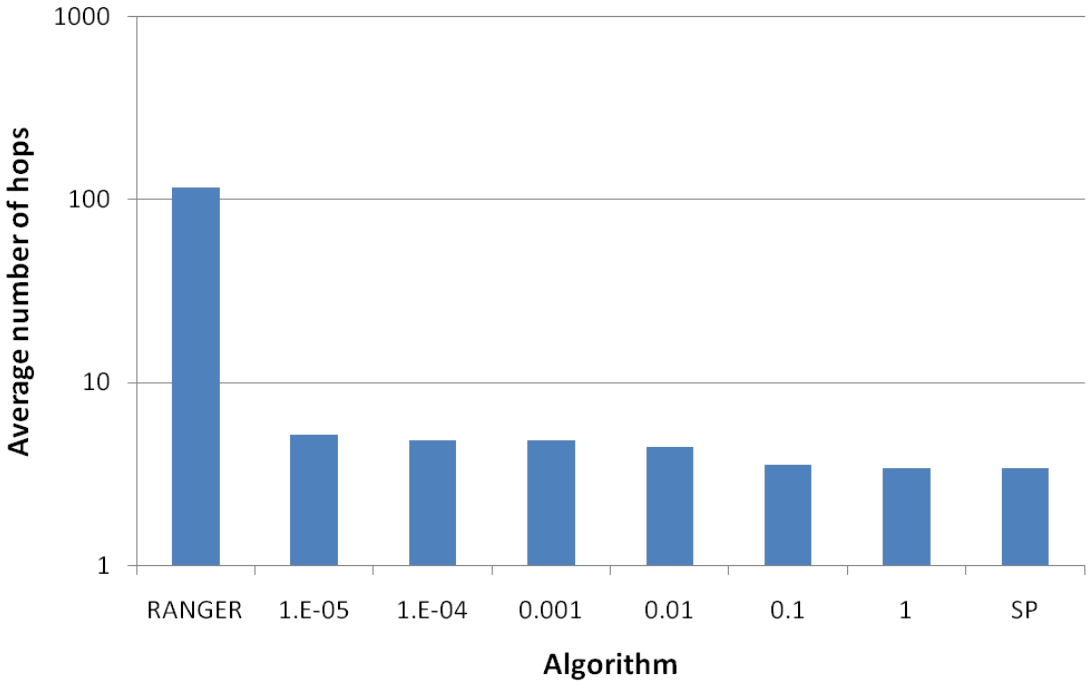


Figure 4.3: Length of solution pathways for RANGER, shortest paths (SP), and LP 2 with varying communication cost.

.

Chapter 5

Flow Allocation in Adversarial Environments: Non-Zero-Sum Games

In many domains, playing a strategy incurs a cost to the player that depends only on the strategy he played. For example, in an urban robot scenario, robots must expend limited energy or fuel in moving along paths, while the cost of deploying a camera for surveillance imposes an attack cost on the adversary. In this chapter we extend the game to consider these kinds of costs, using these paths costs and attack costs as concrete examples. We prove that these kinds of non-zero-sum games are similar to zero-sum games where these costs are factored into both players' payoffs, allowing us to extend our linear programming algorithm to finding equilibria in the non-zero sum games as well. We also consider several further variations on the basic game with multiple sinks and groups of source nodes.

A summary of the notation used in this chapter is provided in Table 5.1. Portions of the work in this chapter was originally published in Okamoto, et al. [46].

5.1 Network Flow Game with Costs (NFG)

We start with the network flow game with costs. This is identical to the zero-sum network flow game described in the previous section, but each player now incurs a cost that depends only on his own strategy, which is subtracted from his payoff. Thus we have the following

b_s	source requirement of $s \in S$
$C_{\text{adversary}}(q)$	cost to the adversary of playing q
$C_{\text{sender}}(f)$	cost to the sender for playing f
c	attack cost vector for the adversary, with c_a being the cost of playing attack $a \in A$
E	set of edges
f	sender strategy, a flow with f^s_{uv} being the amount of flow on edge $(u, v) \in E$ for source node $s \in S$
k	number of attacks that the adversary can play simultaneously
$G = (V, E)$	network with nodes V and edges E
m	number of edges
M	harm matrix with M_{ij} the harm from sending 1 unit of flow on $e_j \in E$ when the adversary plays $a_i \in A$
n	number of nodes
q	an adversary mixed strategy, often expressed as a vector of marginal probabilities over A
S	set of source nodes
$\mathfrak{S} = \{S_1, \ldots, S_K\}$	set of groups of source nodes that partitions the source nodes
T	set of sink nodes
t	a sink node
V	set of nodes
w_e	environmental per-unit-flow cost to sender of using edge $e \in E$

Table 5.1: Summary of notation for Chapter 5

payoff functions:

$$U_{\text{sender}}(f, q) = -qMf - C_{\text{sender}}(f)$$
$$U_{\text{adversary}}(f, q) = qMf - C_{\text{adversary}}(q)$$

where $C_{\text{sender}}(\cdot)$ and $C_{\text{adversary}}(\cdot)$ are the cost functions for the sender and adversary, respectively.

As concrete examples of these cost functions, we consider path costs for the sender and attack costs for the adversary. The path costs represent things such as expenditure of energy for robot movement or communication in a multi-hop sensor network. These costs reduce the payoff for the sender but do not affect the payoff to the adversary. Note that this differs from the costly transmission harm matrix of Section 4.2, which is applicable when the adversary *does* explicitly care about the path costs of the sender. Path costs are computed in the usual way, by associating each edge $e \in E$ with a weight w_e representing the cost of using that edge, and summing the weights of edges in each path. By representing the weights as a $1 \times m$ vector, we can express the sender cost function as

$$C_{\text{sender}}(f) = wf.$$

The adversary suffers attack costs that represent things such as the monetary cost of deploying cameras to conduct surveillance in an area or the cost in resources to conduct an ambush. These costs reduce the payoff of the adversary and are incurred irrespective of whether the attack causes the sender any harm; this may cause the adversary not to utilize some attacks if the costs exceed the benefits. We represent the costs by a $|A| \times 1$ vector c where c_a is the cost of the adversary executing attack a. If we assume the costs are additive for multiple attacks (as in the case of deployment costs of cameras, for instance), we can represent the adversary's cost function as

$$C_{\text{adversary}}(q) = qc.$$

To illustrate the importance of the attack costs on the Nash equilibrium, consider the network in Figure 5.1. Assume that $b_s = 1$ and $k = 1$ and that the adversary can choose to attack the top path or the bottom path with harm matrix

$$M = \begin{bmatrix} 102 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{bmatrix}$$

Let f_i denote the amount of flow on edge e_i and note that the sender's strategy is fully

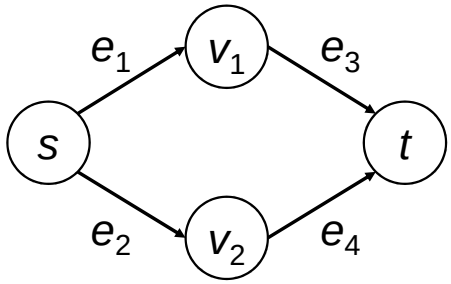


Figure 5.1: An example of a network with two possible paths.

specified by f_1 as $f_2 = 1 - f_1$. Let q_1 and q_2 denote the probability of attacking the top and bottom paths respectively.

When attack costs are zero, the adversary never has incentive not to attack, so $q_1 + q_2 = 1$. In equilibrium the adversary is indifferent between the two attacks so $102f_1 = 3(1 - f_1)$ and so $f_1 = 3/105$. Similarly, the sender is indifferent between the two paths so $102q_1 = 3(1 - q_1)$ and so $q_1 = 3/105$. An intuitive interpretation is that the sender sends most of his flow on the bottom path because of the lower potential for harm, while the adversary, being able to deduce this, attacks the bottom path with high probability because that's where most of the flow is.

Now suppose that attacking the top path has a cost $c_1 = 100$. The adversary's equilibrium strategy remains the same because the sender's payoff hasn't changed, but now for the adversary to be indifferent it must be that $102f_1 - 100 = 3f_2$ so that $f_1 = 103/105$. An intuitive explanation is that the adversary attacks the top path with low probability because of the high attack cost, and the sender, deducing this, sends most of the flow on the top path despite the high potential for harm because the adversary is unlikely to attack there.

The attack costs can also cause the adversary to not execute his maximum number of attacks because the cost outweighs the harm. For example, if $c_1 > 102$ then the sender can set $f_1 = 1$ and a best response by the adversary is to choose $q_1 = q_2 = 0$.

5.2 Computing Nash Equilibrium

Finding Nash equilibria in general non-zero-sum games is computationally more expensive than finding equilibria in zero-sum games. In addition, there may be multiple equilibria with different payoffs for both players, which can complicate the matter of choosing a strategy. In this section we show that these concerns do not arise in the network flow

game with strategy costs, because the Nash equilibria in this game correspond to those of the zero-sum game where both payoffs are affected by costs.

To prove this we will make use of the following function that includes the harm and costs to both players:

$$U(f, q) = qMf + C_{\text{sender}} - C_{\text{adversary}} \tag{5.1}$$

and the following lemma:

Lemma 2. *Let q be an adversary's strategy and let f be a sender's strategy. Then*

1. *f is a best response to q if and only if f minimizes $U(f, q)$.*
2. *q is a best response to f if and only if q maximizes $U(f, q)$.*

Proof. We start with claim 1. By definition, f is a best response to q if and only if f maximizes $U_{\text{sender}}(f, q)$. This happens if and only if f minimizes $-U_{\text{sender}}(f, q) + \alpha$ for any α that is constant (with respect to f). In particular, f is a best response to q if and only if it minimizes $U_{\text{sender}}(f, q) - C_{\text{adversary}}(q) = qMf + C_{\text{sender}}(f) - C_{\text{adversary}}(q) = U(f, q)$.

The proof of claim 2 is similar. □

We can now prove the theorem:

Theorem 2. *(f, q) is a Nash equilibrium for the network flow game with strategy costs if and only if f minimizes $\max_{q'} U(f, q')$ and q maximizes $\min_{f'} U(f', q)$.*

Proof. We start with the backward direction. Assume that f minimizes $\max_{q'} U(f, q')$ and q maximizes $\min_{f'} U(f', q)$. We wish to show that (f, q) is a Nash equilibrium for the network flow game with strategy costs. To see this, observe that f is a minimax strategy in a zero-sum game where the payoff is determined by $U(\cdot)$, and q is a maximin strategy in the same game. Thus, by the Minimax Theorem, f maximizes $U(f, q)$ and q minimizes $U(f, q)$. Thus by Lemma 2, f and q are mutual best responses in the network flow game with strategy costs, and hence (f, q) is a Nash equilibrium for the network flow game with strategy costs.

We proceed next to the forward direction. Suppose that (f, q) is a Nash equilibrium for the network flow game with strategy costs. We prove that f must minimize the maximum $U(f, q')$ by contradiction. Suppose that f is not a minimax strategy and let f' be a minimax strategy. Then there exists a marginal probability vector q'' such that for all marginal

probability vectors q', $U(f', q') < U(f, q'')$. In particular, for $q' = q$, we get

$$\begin{aligned} U(f', q) &= qMf' + C_{\text{sender}}(f') - C_{\text{adversary}}(q) \\ &< q''Mf + C_{\text{sender}}(f) - C_{\text{adversary}}(q'') \\ &\leq qMf + C_{\text{sender}}(f) - C_{\text{adversary}}(q) \qquad \text{because } q \text{ is a best response to } f \\ &= U(f, q) \end{aligned}$$

But $qMf' + C_{\text{sender}}(f') - C_{\text{adversary}}(q) < qMf + C_{\text{sender}}(f) - C_{\text{adversary}}(q)$ implies that $U_{\text{sender}}(f', q) = qMf' + C_{\text{sender}}(f') < qMf + C_{\text{sender}}(f) = U_{\text{sender}}(f, q)$, which means that f is not a best response to q for the sender, contradicting (f, q) being a Nash equilibrium for the network flow game with strategy costs. Hence f must minimize $\max_{q'} U(f, q')$.

Assume now for the purposes of contradiction that q is not a maximin strategy. Let $u = \min_{f'} U(f', q)$ and $f'' \in \arg\min_{f'} U(f', q)$. Let q' be a maximin strategy, that is, $q' \in \arg\max_{q''} \min_{f'} U(f', q'')$, and let $u' = \min_{f'} U(f', q')$. Because q' is maximin and q is not, it follows that $u' > u$. Note that by definition $U(f', q') \geq u'$ and $U(f', q) \geq u$ for all f', and in particular for $f' = f$ we get that $U(f, q') \geq u' > u = U(f'', q)$. Thus,

$$\begin{aligned} U(f, q') &= q'Mf + C_{\text{sender}}(f) - C_{\text{adversary}}(q') \\ &> qMf'' + C_{\text{sender}}(f'') - C_{\text{adversary}}(q) \\ &\geq qMf + C_{\text{sender}}(f) - C_{\text{adversary}}(q) \qquad \text{because f is a best response to q} \\ &= U(f, q) \end{aligned}$$

But this implies that $q'Mf - C_{\text{adversary}}(q') > qMf - C_{\text{adversary}}(q)$, that is, that $U_{\text{adversary}}(f, q') > U_{\text{adversary}}(f, q)$. This contradicts the assumption that q is a best response to f in the network flow game with strategy costs. Thus q must be a maximin strategy. □

Because of Theorem 2, finding an equilibrium sender strategy reduces to finding a minimax strategy. We can extend the network flow LP we used in the previous section to get the linear program LP 4. This LP differs from LP 2 in that the costs are explicitly factored in to objective function. The attack costs are subtracted from the harm that would be inflicted by an attack in Eq. 5.3, reflecting the fact that the adversary's payoff is reduced by the attack cost. The effect of edge costs on the sender is directly represented by adding the edge costs to the objective function that is being minimized.

LP 4 Equilibrium Sender Strategy with Costs

Input: G, b, t, M, c, k
Output: f, H, λ

$$\underset{f,H,\lambda}{\text{Minimize}}\ kH + \sum_{a \in A} \lambda_a + wf \tag{5.2}$$

subject to:

$$H \geq \text{row}_a[M]f - c_a - \lambda_a \qquad \forall a \in A \tag{5.3}$$

$$b_v + \sum_{(u,v) \in E} f_{uv} = \sum_{(v,u) \in E} f_{vu} \qquad \forall v \in V \setminus \{t\} \tag{5.4}$$

$$f \geq 0 \tag{5.5}$$

$$\lambda \geq 0 \tag{5.6}$$

5.2.1 Computing the Adversary's Strategy

We can compute an equilibrium strategy for the adversary using LP 5, which is the dual of LP 4. The q variables represent the adversary's mixed strategy in the usual way. The variable r_v represents the minimum per-unit-flow harm plus edge costs that the sender would suffer if he played optimally against q while starting from $v \in V$. The main constraint is in Equation (5.8): for an edge (u, v), the sender can do no worse from u than he does from v plus the harm and edge costs of sending flow across (u, v). The objective in Equation (5.7) is to maximize the total harm and costs (weighted by the amount of flow) from each source node, less the attack costs. By Theorem 2 this finds an equilibrium strategy for the adversary in the NFG.

However, it only finds *an* equilibrium strategy. What if we want to find an equilibrium strategy for the that gives the sender the worst payoff?

Let a solution to LP 5 be (q^*, r^*) with objective function value X^*. We know that any other solution must also have objective function value X^*, but this may result in different payoffs to the sender in the non-zero-sum NFG. We want to find an equilibrium strategy for the adversary that yields the worst payoff to the sender in NFG. This worst case equilibrium strategy depends on the sender's strategy f, which we previously computed with LP 4.

We use LP 6, which maximizes the harm to the sender while ensuring that all constraints of LP 5 are satisfied and that the solution would also be optimal for LP 5.

Let a solution to LP 6 be q^{worst}, r^{worst}. Things to note:

LP 5 Equilibrium Adversary Strategy with Costs

Input: G, b, t, M, c, w, k
Output: q, r

$$\underset{q,r}{\text{Maximize}} \left(\sum_{s \in S} b_s r_s \right) - \sum_{a \in A} c_a q_a \tag{5.7}$$

subject to:

$$r_u \leq q^T \text{col}_{(u,v)}[M] + r_v + w_{(u,v)} \qquad \forall (u,v) \in E \tag{5.8}$$

$$r_t = 0 \tag{5.9}$$

$$q_a \leq 1 \qquad \forall a \in A \tag{5.10}$$

$$\sum_{a \in A} q_a \leq k \tag{5.11}$$

$$q, r \geq 0 \tag{5.12}$$

LP 6 Worst Case Equilibrium Adversary Strategy with Costs

Input: G, b, t, M, c, w, k, X^*, f
Output: q, r

$$\underset{q,r}{\text{Maximize}}\ qMf \tag{5.13}$$

subject to:

$$X^* = \left(\sum_{s \in S} b_s r_s \right) - \sum_{a \in A} c_a q_a \tag{5.14}$$

$$-q^T \text{col}_{(u,v)}[M] + r_u - r_v \leq w_{(u,v)} \qquad \forall (u,v) \in E \tag{5.15}$$

$$r_t = 0 \tag{5.16}$$

$$q_a \leq 1 \qquad \forall a \in A \tag{5.17}$$

$$\sum_{a \in A} q_a \leq k \tag{5.18}$$

$$q, r \geq 0 \tag{5.19}$$

- q^{worst}, r^{worst} is also a solution to LP 5. (Any feasible solution to LP 6 is a feasible solution to LP 5, and by Eq (5.14), q^{worst}, r^{worst} maximizes the objective function of LP 5.)
- LP 6 always has a solution. (LP 5 always has a solution, so LP 6 must have a solution.)
- Although the sender's payoff in the NFG includes transmission costs, this is not included in the objective function of LP 6 as the term is wf which is constant in this program. The weights w do appear in the constraints.

Let EQ_{adv} be the set of equilibrium adversary strategies. Let $WEQ_{adv}(f)$ be the set of worst case equilibrium adversary strategies for an equilibrium sender strategy f. That is

$$WEQ_{adv}(f) = \{q \in EQ_{adv} : qMf = \max_{q' \in EQ_{adv}} q'Mf\} \tag{5.20}$$

Note that $WEQ_{adv}(f)$ is non-empty.

Theorem 3. *LP 6 computes a worst case equilibrium adversary response to f.*

We can show that this is also a worst case for all equilibrium sender strategies, not just the particular choice of f. Let EQ_{send} be the set of equilibrium sender strategies. We start by noting the interchangeability of equilibria in the zero-sum game related to NFG:

Lemma 3. *Let $f \in EQ_{send}$ and $q \in EQ_{adv}$. Then (f, q) is an equilibrium strategy for the zero-sum game with payoff function $U(f, q) = qMf + C_{sender} - C_{adversary}$ for the adversary and $-U(f, q)$ for the sender.*

Proof. From the minimax theorem. □

We then use this to show the interchangeability of equilibria in NFG:

Lemma 4 (Interchangeability of equilibria of NFG)**.** *Let $f \in EQ_{send}$ and $q \in EQ_{adv}$. Then (f, q) is an equilibrium strategy for the non-zero-sum NFG.*

Proof. From Lemma 3 and Theorem 2. □

We are now ready to prove that the adversary's worst case equilibrium strategies for NFG do not depend on the specific equilibrium strategy played by the sender.

Theorem 4. *Let $f, f' \in EQ_{send}$. Then $WEQ_{adv}(f) = WEQ_{adv}(f')$.*

Proof. We prove this by contradiction. Let $q \in WEQ_{adv}(f)$ and $q' \in WEQ_{adv}(f')$. Assume that $q \notin WEQ_{adv}(f')$.

By interchangeability, (f, q), (f', q), (f', q'), and (f, q') are all equilibria. Thus, $U_{\text{sender}}(f, q) = U_{\text{sender}}(f', q)$ and $U_{\text{sender}}(f, q') = U_{\text{sender}}(f', q')$ so that the sender is indifferent between f and f' when the adversary plays q or q'. Thus we get the following system:

$$\begin{cases} -q'Mf' - C_{\text{sender}}(f') = -q'Mf - C_{\text{sender}}(f) \\ -qMf' - C_{\text{sender}}(f') = -qMf - C_{\text{sender}}(f) \end{cases} \tag{5.21}$$

$$\implies -q'Mf' + qMf' = -q'Mf + qMf \tag{5.22}$$

Because $q \notin WEQ_{adv}(f')$, it must be that $qMf' < q'Mf'$. Thus with Eq (5.22) we get

$$-q'Mf + qMf = -q'Mf' + qMf' < -q'Mf' + q'Mf' = 0 \tag{5.23}$$

$$\implies qMf < q'Mf \tag{5.24}$$

Because $q \in EQ_{adv}(f)$, it follows that $q'Mf \leq qMf$, and thus with Eq (5.24) we get the contradiction that $qMf < qMf$.

Therefore $q \in WEQ_{adv}(f')$ and so $WEQ_{adv}(f) \subseteq WEQ_{adv}(f')$. By symmetry it follows that $WEQ_{adv}(f') \subseteq WEQ_{adv}(f)$ and so $WEQ_{adv}(f) = WEQ_{adv}(f')$. □

By Theorem 4 and Theorem 3, we have shown that LP 6 finds an equilibrium strategy for the adversary that yields the worst payoff for the sender in NFG.

5.3 NFG Example Problem

In this section we work through an example of modeling a real world problem as a network flow game with costs. In this example we consider the problem of police officers in Kabul, Afghanistan responding to an incident at the Hotel Inter-Continental Kabul, one of Kabul's best known hotels and among the most frequently visited by foreigners, especially westerners. On June 28, 2011, it was the target of a 6-hour-long attack by armed terrorists that left at least 21 dead, including all 9 attackers, after a siege by Afghanistan police and NATO forces [55]. In a future incident, terrorists may try to delay the arrival of security forces by obstructing roads leading to the hotel. These roadblocks must be placed in a limited amount of time as they must be set up before the police use the roads, but erecting them too early may tip off the security forces to the impending attack on the hotel.

We model this situation as a network flow game with the sender representing the police and the adversary representing the terrorists trying to delay the arrival of the police at the

Figure 5.2: Graph of Kabul streets near the Hotel Inter-Continental Kabul.

hotel. The sender chooses paths through city streets for the police to take from their police stations to the hotel. The adversary chooses roads for the terrorists to obstruct to delay the police. Thus the adversary "attacks" edges (corresponding to roadblocks on individual roads), and harm is calculated using a form of path intersection harm matrix where the sender incurs harm if and only if the police use a road that has been obstructed by the terrorists. We assume that the terrorists start from a known location, thus restricting which roads they can obstruct in the amount of time they have available.

Figure 5.2 shows the graph of the roads in a roughly 6.5 km by 6.5 km section of western Kabul, constructed from GIS data from the Metro Extracts [1] of the OpenStreetMap database. This graph has 3655 nodes and 8704 edges. The square near the center of the graph depicts the location of the Hotel Inter-Continental Kabul and is the sink node for the network flow game. The upper and lower squares depict the locations of the two nearest police stations and are the source nodes for the network flow game. The last square, on the right, is on a major road running through the area and is the entry point into the region for the terrorists. We assume that the terrorists have time to travel up to 5 km along city streets to place obstructions, and that they do not place roadblocks within 1 km of the hotel or the police stations, as these are the most securely patrolled locations. This limits the adversary to choosing from 5324 edges to attack, shown in bold in Figure 5.2. Attack costs to represent the risk of detection and the use of men and materials in placing roadblocks, with the attack costs proportional to the distance the terrorists must travel to reach the attacked edge from their starting location.

We consider three ways of representing and solving this problem as a network flow game, using different models for the adversary. The first, the NFG approach, directly uses the formulation developed in Section 5.2, with the adversary's pure strategies being any subset of up to k attacks on edges within range, and seeking to maximize his expected payoff of the harm minus the attack costs. In the second approach, the budgeted NFG approach, the attack costs do not factor into the adversary's payoff, but he is instead provided with a budget that he must meet in expectation. The third approach, the normal-form approach, also uses a budget but the adversary must satisfy the stronger condition of meeting the budget with every pure strategy, where the attacked edges in a pure strategy must lie along a single path. We describe these approaches in more detail next.

NFG approach. This approach directly uses the approach from Section 5.2, calculating the sender's equilibrium strategy using LP 4 and the adversary's equilibrium strategy using LP 5. The adversary's choice of strategies is constrained by the number of edges k that he can simultaneously attack and by the costs of the attacks, which are deducted from his payoff. These reflect a situation where there are k terrorists at the starting location, each

[1] http://metro.teczno.com/

LP 7 Budgeted NFG: Equilibrium Adversary Strategy with Budgeted Costs

Input: G, b, t, M, c, w, k, X^*, f
Output: q, r

$$\underset{q,r}{\text{Maximize}} \sum_{s \in S} b_s r_s \tag{5.25}$$

subject to:

$$r_u \geq q^T \text{col}_{(u,v)}[M] + r_v + w_{(u,v)} \qquad \forall (u,v) \in E \tag{5.26}$$

$$r_t = 0 \tag{5.27}$$

$$q_a \leq 1 \qquad \forall a \in A \tag{5.28}$$

$$\sum_{a \in A} q_a \leq k \tag{5.29}$$

$$\sum_{a \in A} c_a q_a \leq C \tag{5.30}$$

$$q, r \geq 0 \tag{5.31}$$

one capable of deploying a single road block, and the terrorists collectively choose roads to obstruct on order by balancing the expected travel time of the police with the costs of erecting road blocks.

Budgeted NFG approach. In this approach, the attack costs do not directly factor into the adversary's payoff. Instead, the adversary has a fixed budget C which he uses to cover attack costs. We solve this problem using a modified version of the NFG approach. The adversary's equilibrium strategy is computed using LP 7. This is identical to LP 5 except that the expected attack costs have been removed from the objective (Equation (5.25)), and there is a new constraint requiring the expected attack costs to be no more than the budget (Equation (5.30)). Note that this only guarantees that the budget is met in expectation, which means that some pure strategies selected from a mixed strategy may exceed the budget, even if the mixed strategy itself meets the budget constraint.

The sender's equilibrium strategy is computed using the dual of LP 7, shown in LP 8. This LP has a new variable X that corresponds to the budget constraint of Equation 5.30 in the adversary's LP.

Normal-Form approach. In this approach a single terrorist vehicle begins at the starting location and moves through the road network, placing up to k road blocks in the desired locations. The attack costs, which are proportional to the distance the terrorists must

LP 8 Budgeted NFG: Equilibrium Sender Strategy with Budgeted Costs

Input: G, b, t, M, c, k
Output: f, H, λ

$$\underset{f,H,\lambda,X}{\text{Minimize}}\ kH + \sum_{a \in A} \lambda_a + wf + XC \tag{5.32}$$

subject to:

$$H \geq \text{row}_a[M]f - \lambda_a - c_a X \qquad \forall a \in A \tag{5.33}$$

$$b_v + \sum_{(u,v) \in E} f_{uv} = \sum_{(v,u) \in E} f_{vu} \qquad \forall v \in V \setminus \{t\} \tag{5.34}$$

$$f \geq 0 \tag{5.35}$$

$$\lambda \geq 0 \tag{5.36}$$

$$X \geq 0 \tag{5.37}$$

travel, depends on the specific path taken by the terrorists to reach the blockaded roads. In addition, we require that the budget constraint is met by every pure strategy, not just in expectation for mixed strategies.

As a result of these assumptions, not all subsets of A of size at most k are feasible pure strategies for the adversary. For example, two edges may be individually within range of the terrorists' starting location, but are too far for the terrorists to blockade both of them while staying under budget. These constraints cannot be represented compactly using the NFG approach of relying solely on the marginal probabilities q_a of each attack $a \in A$. Instead we use the more general but less efficient normal form representation by explicitly enumerating the subsets of A of size at most k, checking which ones meet the budget constraint to determine the feasible pure strategies, then explicitly computing the payoffs. We then solve this normal form game using a standard minimax LP approach.

This approach uses a strategy space for the adversary that is exponentially larger than the strategy space used by the other two approaches. Because of this, the normal-form approach scales poorly with k.

5.3.1 Comparison of Approaches

We first compare the three approaches for $k = 2$. Figure 5.3 shows the equilibrium solution for the NFG approach. The sender's flows are shown using dark blue lines (near-black lines

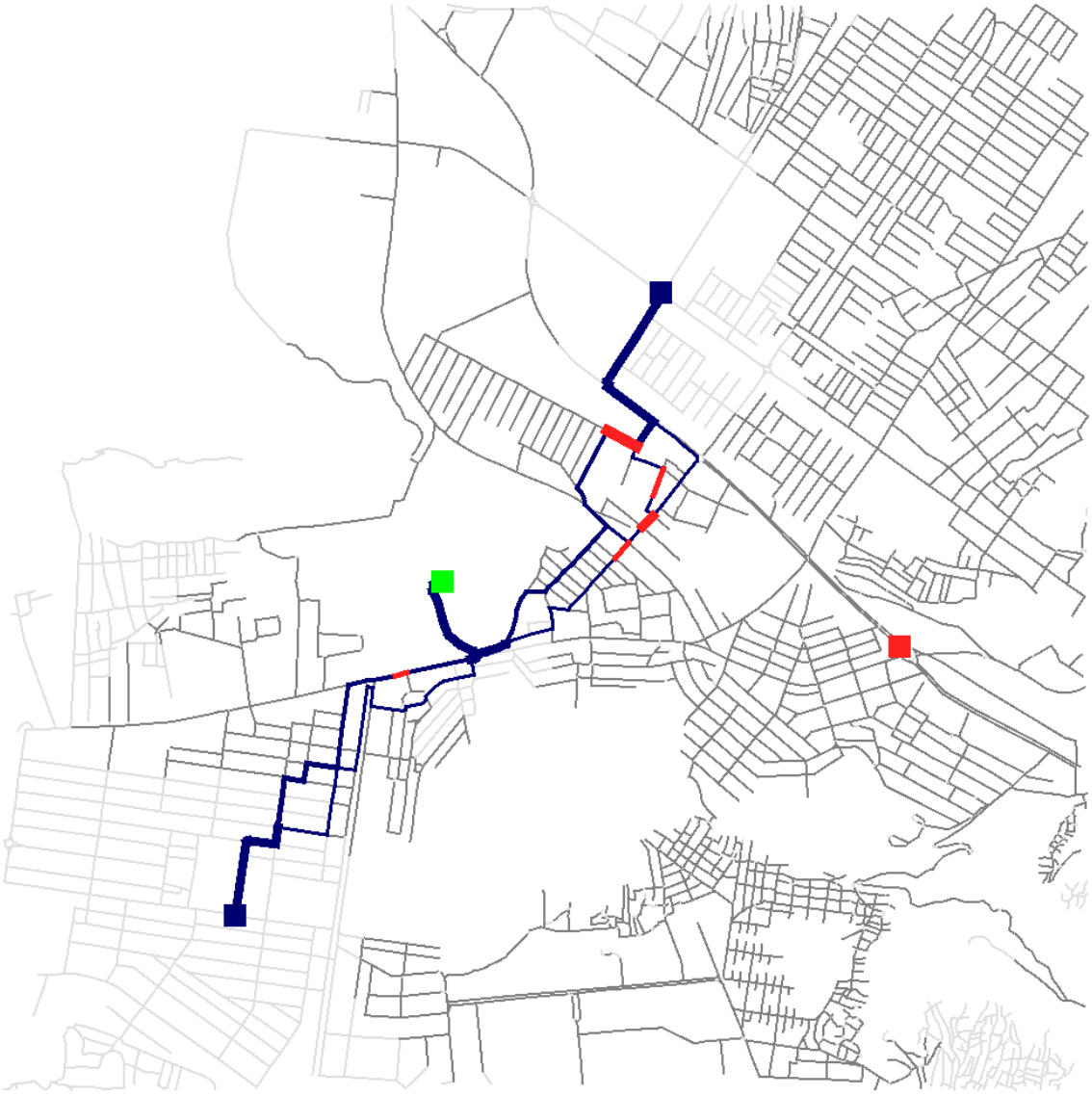


Figure 5.3: Equilibrium solution of NFG approach with $k = 2$.

in black-and-white), with thicker lines indicating a greater amount of flow. The adversary's attacks are shown using red lines (gray lines in black-and-white) on attacked edges, with thicker lines indicating a higher marginal probability of attack. Most of the adversary's attacks involve edges from the northern police station. This is because they are closer to the adversary's starting location and hence cheaper for the adversary, and in the NFG approach the attack costs are deducted from the adversary's payoff. The adversary does attack an edge on a path from the southern police station, but only with low probability.

There is also relatively little branching of the sender's flows because any reduction in harm from the adversary's attacks would not compensate for the increase in travel time from taking longer paths. For example, police from the northern police station could head west before turning south to approach the hotel, joining with the flow from the southern police station and approach the hotel from the west. This would greatly reduce the probability of encountering a terrorist roadblock, but would also greatly increase the travel time due to the longer routes.

Note also that the edges attacked by the adversary form a cut in the flow from the northern police station but not for the southern police station. Because in equilibrium the sender is indifferent between the paths he has chosen to send flow on, this means that the effect of the southern attack is to decrease the value of the shorter path with the attacked edge, to make it equally valuable as the longer path that is not attacked. All paths with flow from the northern police station include at least one edge that is attacked by the adversary with non-zero probability, and the marginal probabilities of attack are balanced to make the sender indifferent.

Figure 5.4 shows the equilibrium solution for the budgeted NFG approach. The main difference from the NFG approach is that the adversary is much more likely to attack edges near the southern police station. This is because the attack costs do not factor into the adversary's payoff. Thus a high cost of attack will not dissuade the adversary from playing that attack if there remaining budget left and no better alternative.

Figure 5.5 shows the equilibrium solution for the normal-form approach. The result for the adversary is similar to the budgeted NFG approach. Like the budgeted NFG approach and unlike the NFG approach, the attack costs do not directly factor into the adversary's payoff, possibly enabling the adversary to attack edges near the southern police station. However, the strict budget constraint requires that both attacks on edges in a pure adversary strategy must be reachable from the start node in the same path. Thus, when the adversary travels to attack near the southern police station, it chooses to blockade two edges.

We next compare the NFG and budgeted NFG approaches for $k = 10$. Because of its poor scalability, the normal-form approach was not able to be used. Figure 5.6 shows

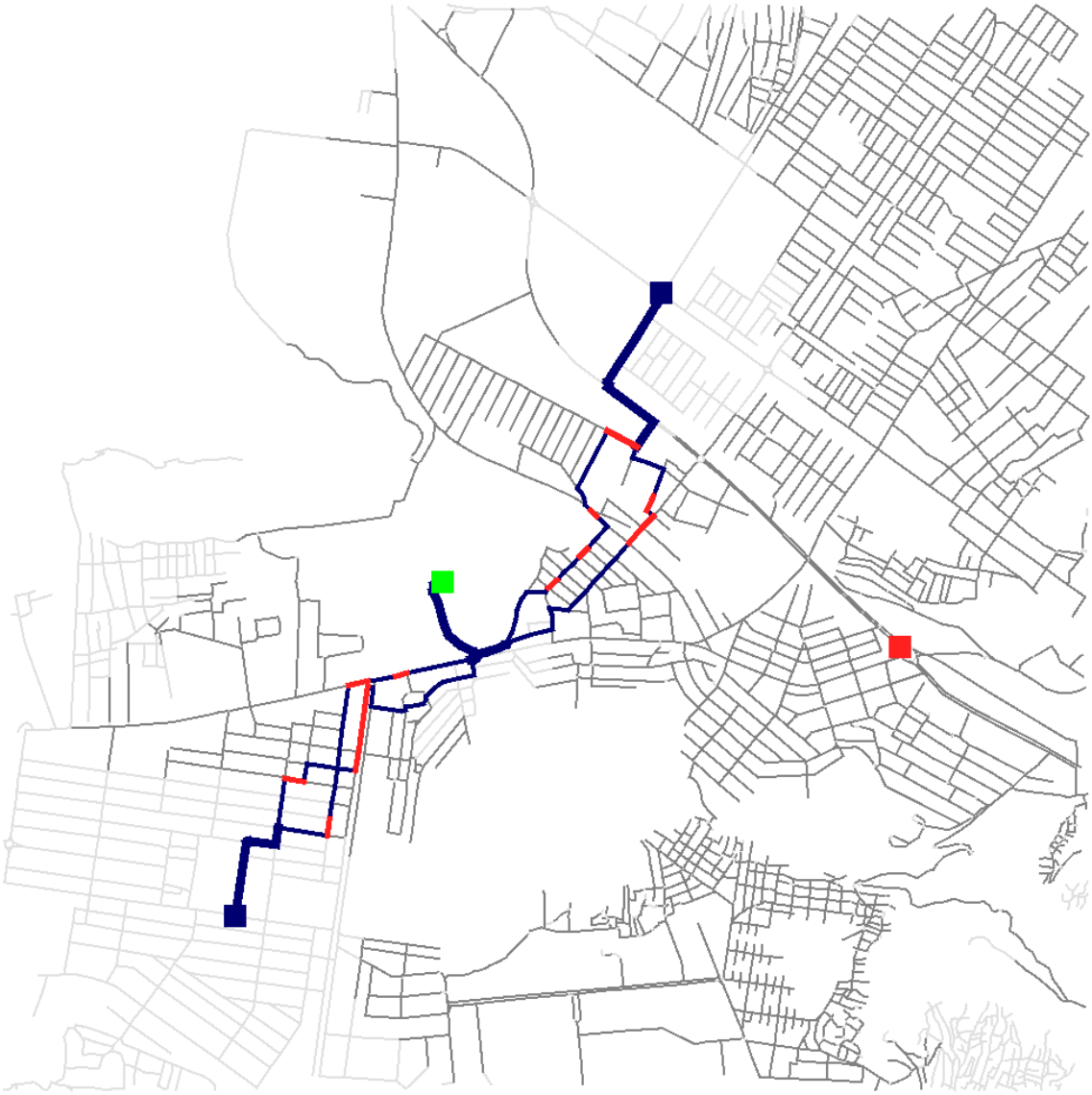


Figure 5.4: Equilibrium solution of budgeted NFG approach with $k = 2$.

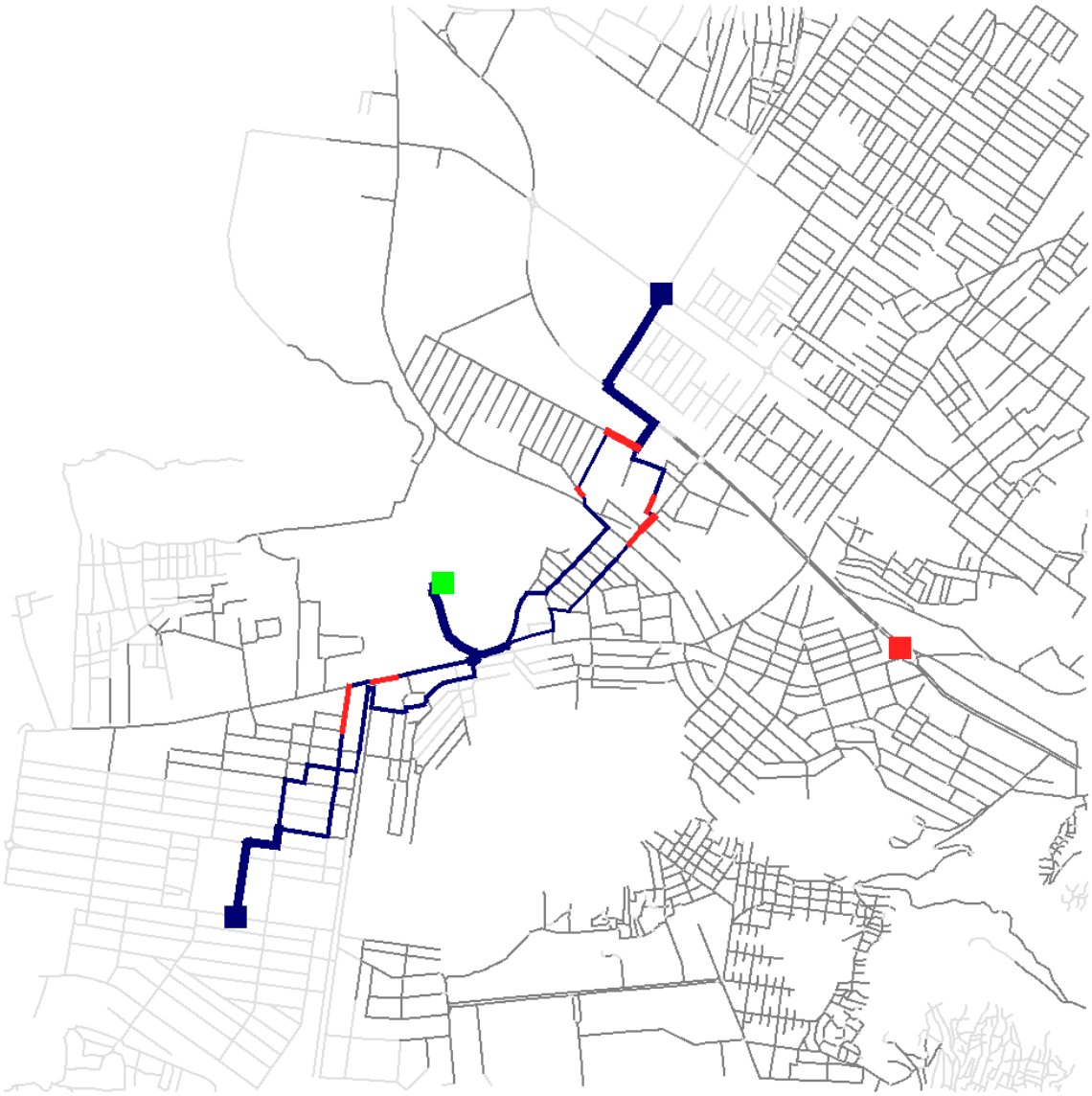
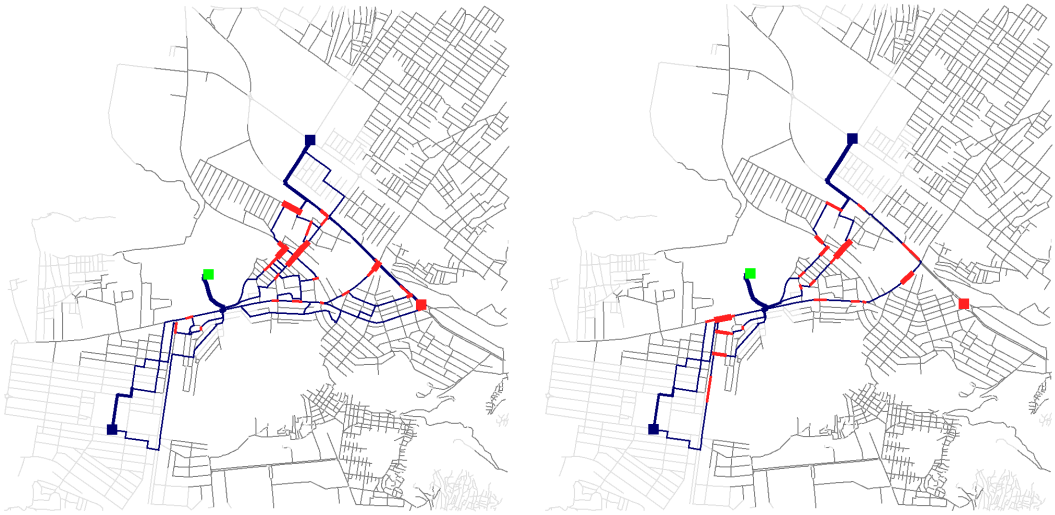


Figure 5.5: Equilibrium solution of normal-form approach with $k = 2$.

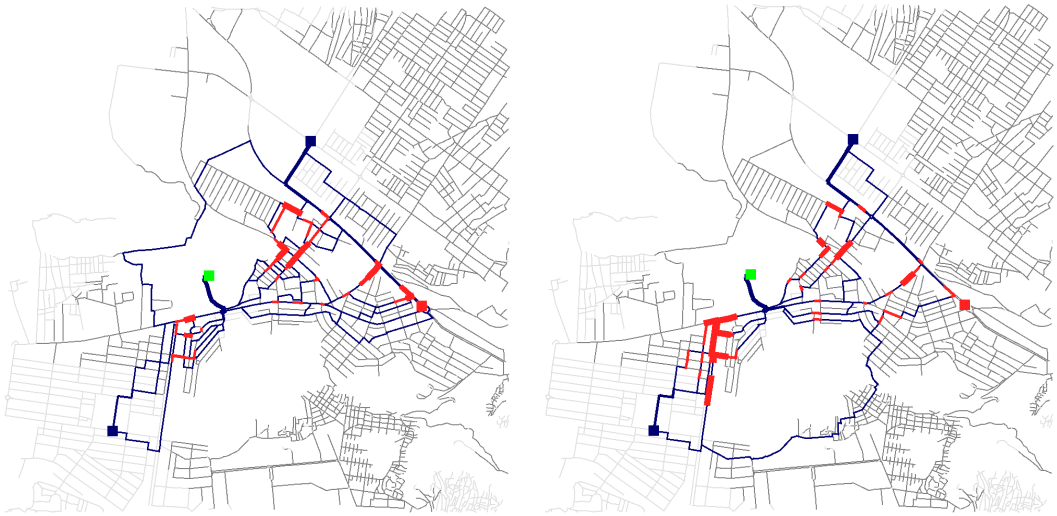


(a) (b)

Figure 5.6: Equilibrium solution of (a) NFG approach and (b) budgeted NFG approach with $k = 10$.

the equilibrium solution for the NFG and budgeted NFG approaches. Compared to the solutions for $k = 2$, the sender's flows are much more extensively branched. This is because the adversary is able to attack five times as many edges as in the $k = 2$ setting, providing a stronger incentive for the sender to reduce the amount of flow on any single edge. Note that as in the $k = 2$ case, the adversary under the budgeted NFG approach is much more likely to attack edges near the southern police station than he is under the NFG approach. The corollary of this is that the adversary is more likely to attack near the northern police station for the NFG approach than for the budgeted NFG approach. As a result, with the NFG approach the sender has greater incentive to reduce the flow on any single edge, leading to even more branching in the flows from the northern police station than seen in the budgeted NFG approach.

These differences are brought out even further for $k = 20$, as shown in Figures 5.7. With the NFG approach, the adversary's attacks in the north provide the sender with incentive to branch further, even using the long counter-clockwise path around the hotel to approach the hotel from the west. Similarly, with the budgeted NFG approach, the adversary's attacks in the south provide the sender with incentive to use a long path to approach the hotel from the east.



(a) (b)

Figure 5.7: Equilibrium solution of (a) NFG approach and (b) budgeted NFG approach with $k = 20$.

5.4 Network Flow Game with Multiple Sinks (NFG-MS)

In the games that we have considered so far there is a single known sink. This models domains like a robot team where there is a single known location (e.g., the location of a joint task) that all robots must travel to, or a sensor network where there is a single base station capable of performing data fusion. However, in many domains there are multiple sinks that the source nodes must chose from: there are multiple joint tasks that the robot team can perform, or multiple base stations that the sensors could transmit their data to.

The network flow game with multiple sinks (NFG-MS) extends the network flow game with costs from a single sink $t \in V \backslash S$ to a set of sinks $T \subset V \backslash S$. For each sink $t \in T$ there is a reward $r_t > 0$ that the sender receives for using that sink. This models heterogeneity between sinks, such as some tasks being more important or some base stations having more battery power or computational power. In the absence of an adversary the optimal behavior for the sender is to choose a sink that maximizes utility (the reward minus the costs). However, this kind of deterministic behavior is vulnerable to attack. Thus, in adversarial environments the agents will have to balance reward, costs, and harm by randomizing over both the sink to use and the paths to the sink. When the flows are interpreted as mixed strategies in the path game, the sender in each pure strategy must choose paths for each

source node to the same sink. This is achieved by requiring that for each sink $t \in T$, the source nodes all send the same fraction of their flow to t. For convenience we assume that the sources all must transmit unit flow, that is, $b_s = 1$ for all $s \in S$.

LP 9 computes an equilibrium sender strategy for the network flow game with multiple sinks. In contrast to the previous LPs which used a single flow approach, this LP uses multicommodity flows, with a separate type of flow for each source. The flow for source $s \in S$ is denoted by f^s. For readability we use placeholder variables to represent the various components of the payoff: R for the expected reward, H for the expected harm (less attack costs), and C^{edge} for the expected edge costs. For each $t \in T$, the variable d_t represents the marginal probability of t being chosen as the sink. The expected reward is thus dr (Eq. (5.39)). The harm is calculated as before except with the sum over all types of flows, as shown in Eqs. 5.40 and 5.41. The edge costs are also calculated as before except with the sum over all types of flows in Eq. 5.42. The flow conservation requirements have been extended to each individual type of flow, and d_t explicitly models the amount of flow that is deposited in sink t. By using a single variable d_t for all types of flow, we ensure that the the source nodes all use t as a sink for the same fraction of their flow.

Note that in the previous network flow LPs, the source nodes were required to transmit all of their flow to the sink. In NFG-MS, the source nodes may end up only sending some of their flow to the sinks. This is reflected in Eq. (5.46), where the sum of the d variables is required to be less than or equal to one, rather than equal to one. This is because in some cases the sum of the expected costs and harm may exceed the expected reward for using a sink.

5.5 Network Flow Game with Multiple Groups (NFG-MG)

In NFG-MS, all source nodes are required to send to the same sink in a pure strategy. This makes sense when the source nodes correspond to a cooperative group like a team of robots performing a joint task or a set of sensors tracking the same object. However, in most domains there are multiple groups: multiple subteams performing different tasks or different sensors tracking different objects.

The network flow game with multiple groups (NFG-MG) extends NFG-MS to handle these cases and is solved (for the sender) by LP 10. The source nodes are partitioned into groups $\mathfrak{S} = \{S_1, \ldots, S_{|\mathfrak{S}|}\}$. The reward, edge costs, and harm matrix are indexed by group, r_t^i, w_e^i, and M^i for all $i \in \{1, \ldots |\mathcal{S}|\}$, $t \in T$, and $e \in E$. All source nodes within a group must send flow to the same sinks in the same proportions. Furthermore, sinks can only be used by a single group at a time; this corresponds to each task only being able to be

LP 9 Equilibrium Sender Strategy with Multiple Sinks

Input: G, b, T, M, c, k
Output: f, H, λ

$$\underset{f,R,H,H',C^{\text{edge}},d,\lambda}{\text{Maximize}} \quad R - H - C^{\text{edge}} \tag{5.38}$$

subject to:

$$R = dr \tag{5.39}$$

$$H = kH' + \sum_{a \in A} \lambda_a \tag{5.40}$$

$$H' \geq \text{row}_a[M](\sum_{s \in S} f^s) - c_a - \lambda_a \qquad \forall a \in A \tag{5.41}$$

$$C^{\text{edge}} = w \sum_{s \in S} f^s \tag{5.42}$$

$$b_v + \sum_{(u,v) \in E} f^s_{uv} \geq \sum_{(v,u) \in E} f^s_{vu} \qquad \forall v \in V \setminus \{t\}, \forall s \in S \tag{5.43}$$

$$\sum_{(u,t) \in E} f^s_{ut} \geq d_t + \sum_{(t,u) \in E} f^s_{tu} \qquad \forall t \in T, \forall s \in S \tag{5.44}$$

$$d_t \leq \sum_{(u,t) \in E} f^s_{ut} \qquad \forall t \in T, \forall s \in S \tag{5.45}$$

$$\sum_{t \in T} d_t = 1 \tag{5.46}$$

$$f \geq 0 \tag{5.47}$$

$$\lambda \geq 0 \tag{5.48}$$

$$d \geq 0 \tag{5.49}$$

performed once, or each base station only able to perform fusion for a single object at a time. This assumption can be relaxed either by modifying the constraints (Eq. (5.59)) of LP 10 or by adding multiple copies of the sink nodes as desired.

5.6 Bayesian Games: Dealing with Uncertainty

In many security applications, it is unrealistic to assume that complete information on the adversary is available because adversaries are hostile and usually secretive. In military and law enforcement domains, intelligence analysts, criminologists, and other experts collect relevant data on real and possible adversaries and develop inherently uncertain estimates of their capabilities and motives. In this section we represent that uncertainty as probability distributions over the maximum number of attacks that they can execute, the harm matrices, and the attack costs. We develop ways to reason strategically over this incomplete information by adopting the Bayesian game framework and find polynomial time algorithms for finding equilibria.

For simplicity, we present our approach in the context of NFG, the single-group, single-sink network flow game with costs solved by LP 4 and LP 5 for the case without uncertainty. It is straightforward (although requiring more elaborate notation) to extend our approach to NFG-MS and NFG-MG.

5.6.1 Uncertain k

In many situations it is not possible for the sender to know the adversary's capabilities with certainty. The sender can act as if he has the full knowledge, but he then might perform badly. For example, suppose that the game is the same as in Figure 5.1, but now $k = 2$. In equilibrium, the adversary strategy is $q_1 = 1/34$ and $q_2 = 1$, and the sender strategy is $q_1 = 100/102$ and $q_2 = 2/102$. The expected harm for the sender will thus be 3. However, if the sender does not know that $k = 2$ now, and continue to play his strategy for $k = 1$, the adversary will exploit it and will always attack v_1 and v_2. The sender's harm will thus increase to 100.116. Therefore, when the sender does not sure about the exact value of k, he will have to estimate it. We represent this by a probability distribution q over possible values of k, which we assume is known to both players. Given this distribution, we formulate the sender's problem as a *Bayesian game*. A Bayesian game is one in which information about characteristics of the other players is incomplete. There is a probability distribution over possible *types* for each player, and the type of a player determines that player's payoff function. In our case, the sender has only one type, and the type of the

LP 10 Equilibrium Sender Strategy with Multiple Groups

Input: G, b, $\mathfrak{S}$, T, M, c, k
Output: f, H, λ

$$\underset{f,R,H,H',C^{\text{edge}},d,\lambda}{\text{Maximize}} \quad R - H - C^{\text{edge}} \tag{5.50}$$

subject to:

$$R = \sum_{i=1}^{|\mathfrak{S}} \sum_{t \in T} d_t^i r_t^i \tag{5.51}$$

$$H = kH' + \sum_{a \in A} \lambda_a \tag{5.52}$$

$$H' \geq \left(\sum_{i=1}^{|\mathfrak{S}|} \text{row}_a[M^i] f^i \right) - c_a - \lambda_a \qquad \forall a \in A \tag{5.53}$$

$$C^{\text{edge}} = \sum_{i=1}^{|\mathfrak{S}|} \sum_{e \in E} w_e^i f_e^i \tag{5.54}$$

$$b_v^s + \sum_{(u,v) \in E} f_{uv}^s \geq \sum_{(v,u) \in E} f_{vu}^s \qquad \forall v \in V \setminus \{t\}, \forall s \in S \tag{5.55}$$

$$\sum_{(u,v) \in E} f_{uv}^s \geq d_t^{\text{group}(s)} + \sum_{(v,u) \in E} f_{vu}^s \qquad \forall v \in T, \forall s \in S \tag{5.56}$$

$$d_t^{\text{group}(s)} \leq \sum_{(u,t) \in E} f_{ut}^s \qquad \forall t \in T, \forall s \in S \tag{5.57}$$

$$\sum_{t \in T} d_t^i = 1 \qquad \forall i \in \{1, \ldots, |\mathfrak{S}|\} \tag{5.58}$$

$$\sum_{i=1}^{|\mathfrak{S}|} d_t^i \leq 1 \qquad \forall t \in T \tag{5.59}$$

$$f \geq 0 \tag{5.60}$$

$$\lambda \geq 0 \tag{5.61}$$

$$d \geq 0 \tag{5.62}$$

LP 11 Equilibrium Sender Strategy in a Bayesian game (uncertain k)

Input: G, M, A, c, k, $\Pr(k)$
Output: f, $\{R^k\}$, $\{\lambda^k\}$

$$\underset{f,\{R^k\},\{\lambda^k\}}{\text{Minimize}} \; wf + \sum_{k=1}^{|A|} \Pr(k) \left(kR^k + \sum_{a \in A} \lambda_a^k \right) \tag{5.63}$$

subject to:

$$R^k \geq \text{row}_a[M]f - c_a - \lambda_a^k \qquad \forall a \in A, \forall k \in [1..|A|] \tag{5.64}$$

$$\sum_{(v,u) \in E} f_{vu} = b_v + \sum_{(u,v) \in E} f_{uv} \qquad \forall v \in V \setminus \{t\} \tag{5.65}$$

$$f_{uv} \geq 0 \qquad \forall (u,v) \in E \tag{5.66}$$

$$\lambda_a^k \geq 0 \qquad \forall a \in A, \forall k \in [1..|A|] \tag{5.67}$$

$$R^k \geq 0 \qquad \forall k \in [1..|A|] \tag{5.68}$$

adversary is determined by the value of k. We denote the probability that the adversary is of type k as $\Pr(k)$.

The sender's optimal equilibrium strategy can be computed using LP 11. This LP is similar to LP 4, except that instead of minimizing maximum adversary expected payoff for a specific value of k, it minimizes the weighted sum of the expected rewards of every possible k, weighted by their possibilities. The number of variables and constraints is still polynomial in n and $|A|$ and so this LP can be solved in polynomial time.

We must verify that in the Bayesian game, minimizing the adversary's maximum expected payoff also maximizes the sender's expected payoff.

Lemma 5. *Let f be a sender strategy, $\{q^k\}$ be an adversary strategy, and $\Pr(k)$ be a probability distribution over values of k. Then f is a best response to $\{q^k\}$ if and only if f minimizes expected reward relative to q and $\Pr(k)$.*

Proof. Forward: Assume f is a best response to $\{q^k\}$. Then f minimizes $\sum_{k=1}^{|Z|} \Pr(k) q^k M f$. Therefore it also minimizes $\sum_{k=1}^{|A|} \Pr(k) q^k M f - \Pr(k) q^k c$ because $\Pr(k) q^k c$ is constant with respect to f. Backward direction is similar. □

Even though the adversary knows his type (i.e., the correct value of k) he cannot use LP **??** to find his equilibrium strategy because the sender does not know the exact value of k. The adversary's equilibrium strategy can instead be computed by the dual to LP 11.

LP 12 Equilibrium Sender Strategy in a Bayesian game (uncertain M and c)

Input: G, $\{M^i\}$, $\{c\}$, k, L, $\Pr(i)$
Output: f, $\{R^i\}$, $\{\lambda^i\}$

$$\underset{f,\{R^i\},\{\lambda^i\}}{\text{Minimize}} \; wf + \sum_{i=1}^{L} \Pr(i) \left(kR^i + \sum_{a \in A} \lambda_a^i \right) \tag{5.69}$$

subject to:

$$R^i \geq \text{row}_a[M^i] f - c_a^i - \lambda_a^i \qquad \forall a \in A, \forall i \in [1..L] \tag{5.70}$$

$$\sum_{(v,u) \in E} f_{vu} = b_v + \sum_{(u,v) \in E} f_{uv} \qquad \forall v \in V \setminus \{t\} \tag{5.71}$$

$$f_{uv} \geq 0 \qquad \forall (u,v) \in E \tag{5.72}$$

$$\lambda_a^i \geq 0 \qquad \forall a \in A, \forall i \in [1..L] \tag{5.73}$$

5.6.2 Uncertain Payoffs

Another way in which the sender may be uncertain of the adversary is by not knowing the payoffs and costs. Suppose instead that he has a probability distribution over L possible payoff matrices and attack costs (types of adversaries). For $i \in [1..L]$, let $\Pr(i)$ be the probability that the adversary is of type i with a harm matrix M^i and cost of attacks c^i. The sender's optimal equilibrium strategy is computed by LP 12. As before, the adversary's strategy can be computed by taking the dual of this LP.

If we assign $\Pr(i) = 1$ for every $i \in [1..L]$ we get a linear program which solves another interesting variant of our problem. Consider a game with one sender and multiple adversaries. The adversaries choose their strategies independently of each other (i.e., no colluding). The adversaries have different harm matrices and costs for attacking nodes and the total harm to the sender is the sum of the harm resulting from each adversary's attack. The payoff to each adversary depends only on his own strategy and the sender's strategy; it does not depend on the strategies of any of the other adversaries. For now, let's assume that every adversary can attack k nodes. By assigning $\Pr(i) = 1$ for every $i \in [1..L]$ we get that LP 12 computes the equilibrium strategy for the sender in the multiple adversaries game too! As for the attacker's equilibrium strategies, we get an interesting observation: since the strategies can be computed by the dual of LP 12, they are in fact correlated. Even though the adversaries choose their strategies independently of each other, due to the strategic consideration they behave as if they coordinate their moves.

5.6.3 Experiments

In this section we experimentally evaluate our algorithms. We empirically evaluated our approach through simulation of two multiagent systems, a multirobot team operating in an urban environment and a sensor network transmitting through a multihop, ad hoc network. Each data point is the average over 20 independent randomly generated instances.

Source nodes in the sensor network were sensor nodes that were generating data (for example, readings from a target) that had to be transmitted back for fusion at one of several base stations (the sink nodes). Other nodes in the network were used as relays between the sensors and the base stations. Nodes were distributed uniformly at random in a 10×10 region of the plane. Edges were added between nodes within a Euclidean distance of 1 of each other, yielding a unit disk graph, a commonly used abstraction of wireless network topologies. This resulted in networks that on average had many paths between any two nodes, but were still not fully connected (in which case the problem is trivial).

The graph in the robot team represented an urban road network, with source nodes representing the starting locations of the robots. We conducted experiments on both synthetic, square grid graphs. We also used topologies extracted from GIS data [2] of the road network of Afghanistan as recorded by the Afghanistan Information Management Service. The graph is shown in Figure 5.8.

Source nodes were selected uniformly at random from among the nodes, and sink nodes were selected uniformly at random from among the nodes excluding the source nodes. For the presented experiments $|T| = 10$ and $k = 10$.

We used a commercial LP solver, Gurobi 4.6.1, running on a 64-bit Windows 7 computers with a 2.8 GHz, second-generation Intel Core processor and 8 GB of RAM.

The first set of experiments examined the impact of varying the number and size of groups. One might expect that having the source nodes divided into more groups would increase the running time as there would be more possible combinations of sinks chosen for each group. To investigate this, we varied the number of groups from one to four while independently varying the size of each group from one to ten. This resulted in instances where the total number of source nodes varied from one to forty.

Figures 5.9 and 5.10 show the average running times plotted for the total number of source nodes, using log-log axes. Note that the points for all groups are roughly collinear. This shows that it is the total number of source nodes, not the specific distribution of the source nodes into groups, that affects running time. Also note that the points in both plots show a linear trend, which because of the log-log plot indicates that the running time increases

[2] http://www.mapcruzin.com/free-afghanistan-roads-arcgis-maps-shapefiles.htm

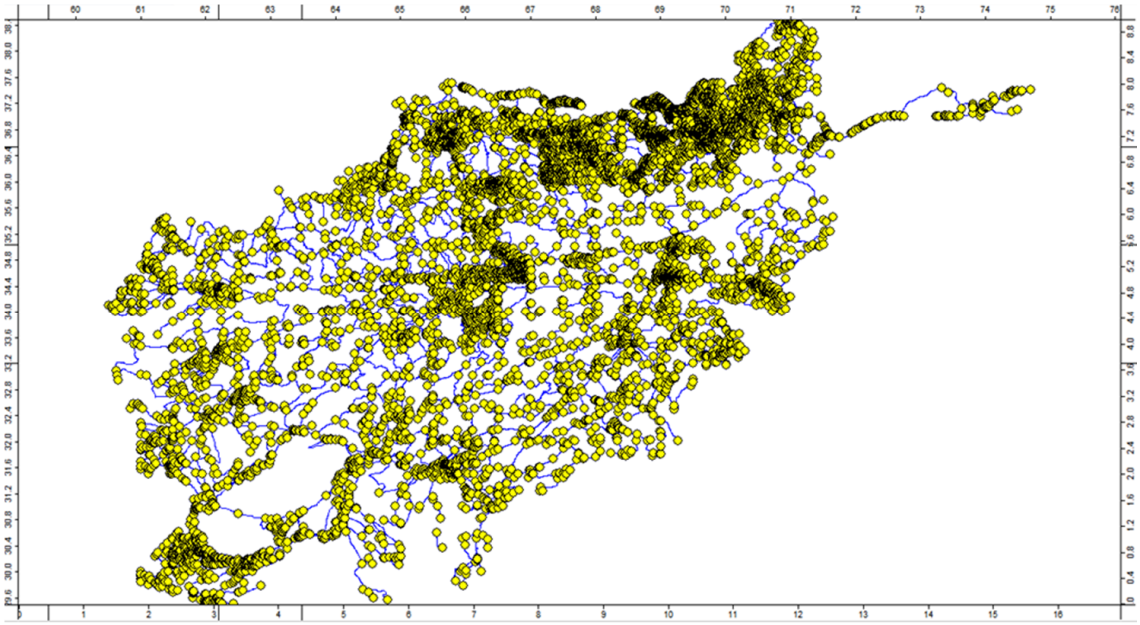


Figure 5.8: Graph of the road network of Afghanistan.

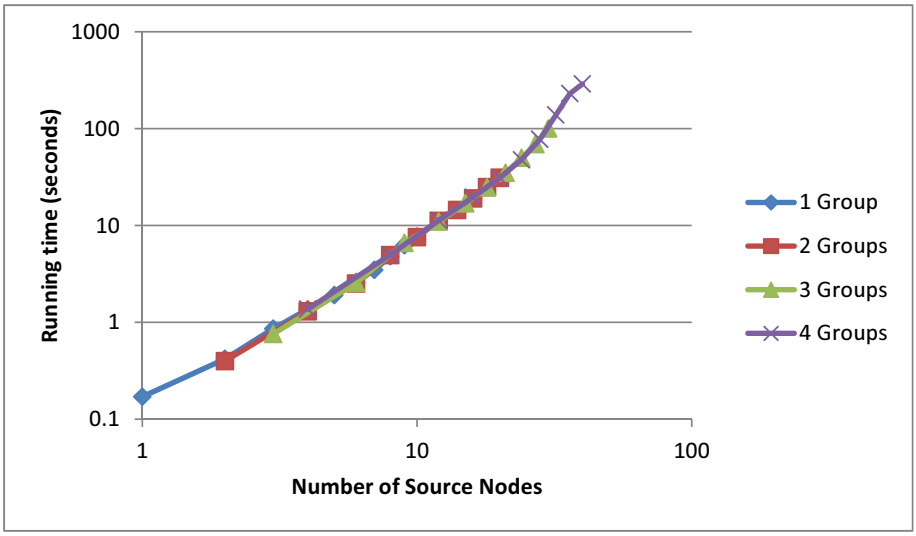


Figure 5.9: Running time of LP 10 vs. number of source nodes as the number of number of groups and number of source nodes per group is varied for the grid network

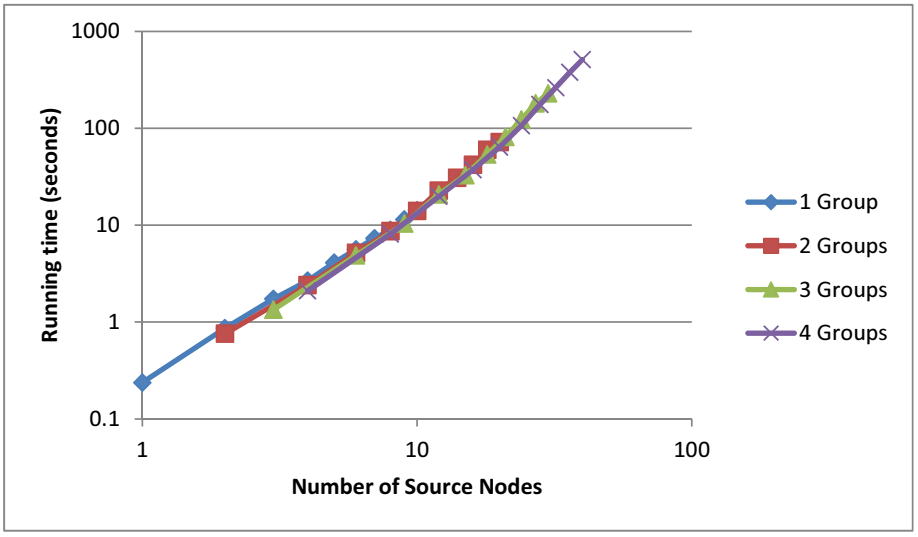


Figure 5.10: Running time of LP 10 vs. number of source nodes as the number of number of groups and number of source nodes per group is varied for the disk network.

polynomially with the number of source nodes, as expected for a linear programming-based approach. For the grid network, the best fit line has a slope of 2.0247 (with $R^2 = 0.9816$) while for the disk network the best fit line has a slope of 2.0945 (with $R^2 = 0.9855$), indicating that the effect of the number of source nodes on running time is close to quadratic.

The next set of experiments looked at the impact of the size of the graph on running time by varying the number of nodes for each type of graph. For the grid graph we varied the size of the graph from 100 to 10,000 nodes. For the disk graph we varied the number of nodes that we placed from 100 to 10,000 nodes as well; however, due to the stochastic nature of network formation, not all nodes were connected in the every resulting instance which could cause problems if source in the same group could not reach the same sink. For each instance we extracted the largest connected component and used this as the graph of the environment. The average number of nodes in these graphs varied from 81 to 9608. For the graph of the road network of Afghanistan, we isolated rectangular geographical regions of varying sizes around Kabul and extract the subgraph induced by the nodes within the rectangular regions. Again this could result in some nodes being disconnected (for example, because their connections in the road network extended outside of the specific rectangular region we were considering) and so we took the largest connected component for each region size as the graph. The number of nodes in the Afghanistan maps varied from 123 to 12,459.

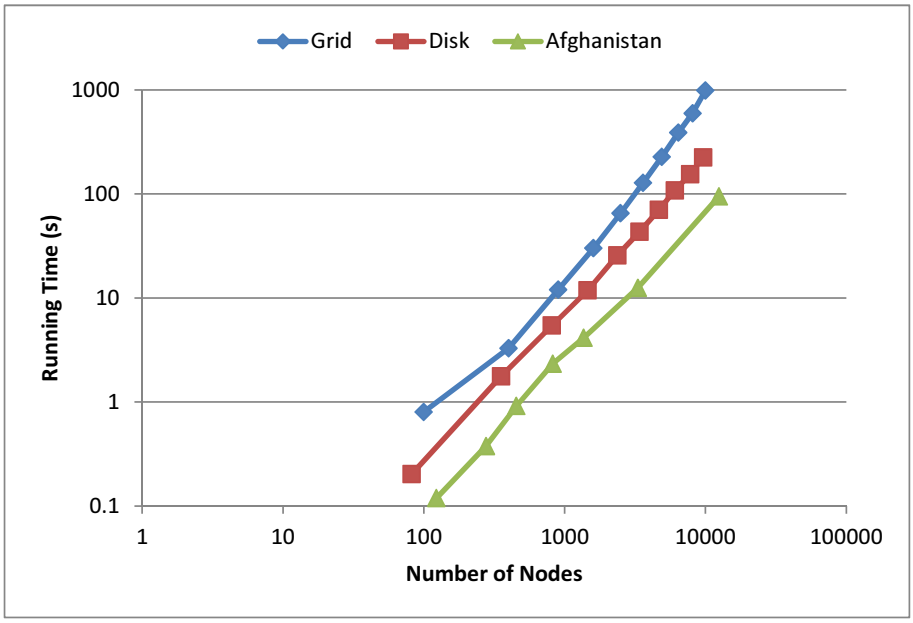


Figure 5.11: Running time of LP 10 vs. number of nodes.

The running times vs. the number of nodes for the three topologies are shown in Figure 5.11 using a log-log plot. The grid network has the highest average running times overall, followed by the disk network, and then the Afghanistan network. The points show a linear trend, with slopes ranging from 1.58 for grid network to 1.42 for the Afghanistan network, indicating polynomial running time that is sub-quadratic. This shows that LP 10 is scalable for large numbers of nodes.

We also plot the running times against the number of edges. This is important as the number of paths through the network depends not just on the number of nodes but also on the number of edges. The results are shown in Figure 5.12, again using logarithmic axes. We again see linear trends, this time with slopes ranging from 1.55 for the grid network to 1.41 for the Afghanistan graph. Surprisingly, the running times on the disk and Afghanistan graphs are roughly collinear, while the running times on the grid network are significantly higher. This is interesting because the disk network has more edges overall, from 90 to 53,893, while the grid network has between 360 and 39,600 edges and the Afghanistan network between 308 and 33,846 edges.

Our final set of experiments compares our approach with two heuristics. Minimum Harm (MH) minimizes the possible harm that can be caused by the adversary but does not consider the sender's environmental costs. Shortest Paths (SP) minimizes the environmental

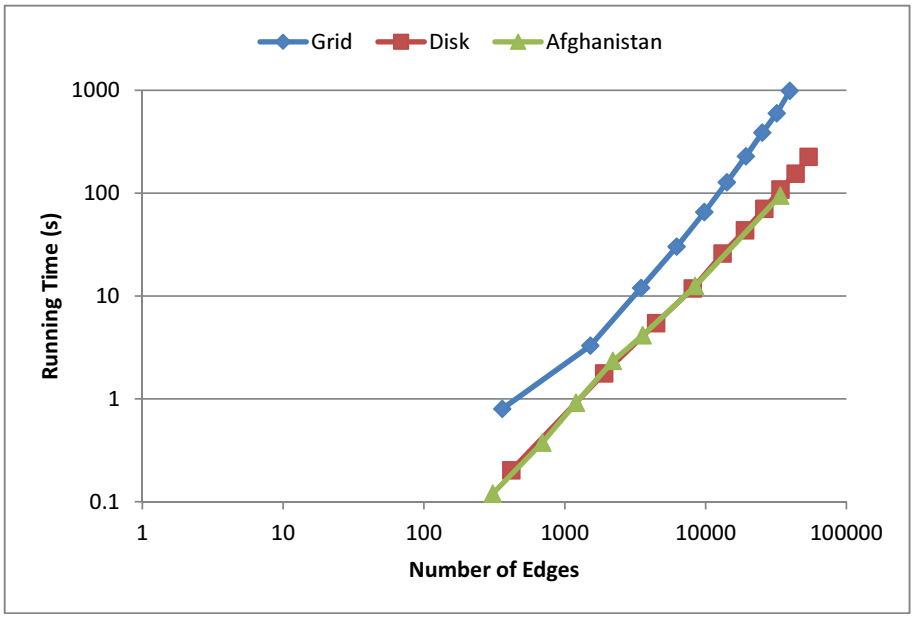


Figure 5.12: Running time of LP 10 vs. number of edges.

cost but ignores any possible attack by an adversary. We vary the relative weight of the harm and environmental costs in the sender's payoff according to the formula

$$U_{\text{sender}}(f, q) = -qMf - \gamma wf, \tag{5.74}$$

where $\gamma > 0$ is a constant sender cost factor that we vary from 2^{-6} to 2^2 in powers of 2.

Figures 5.13 –5.15 shows the total costs to the sender (the sum of the harm and environmental costs) when the adversary plays a worst case equilibrium strategy on different graph topologies. This adversary strategy does not depend on the sender's choice of strategy. Note that the sender plays off-equilibrium when he plays MH and SP, and hence the strategy profiles are not in equilibrium. Indeed, the adversary's strategy may not even be a best response to the sender's strategy, as it was chosen assuming that the sender would play an equilibrium strategy.

From the figures it is clear that the EQ and SP strategies perform similarly, while the MH strategy does increasingly poorly as the sender cost factor increases. This is to be expected, as the MH strategy does not factor in the environmental costs at all, instead choosing to divide the flow over as many paths as possible to reduce exposure to attack, even if those paths are very long. As γ is increased, the environmental costs come to dominate the sender's costs and so MH does poorly. From these results it seems that SP is superior to

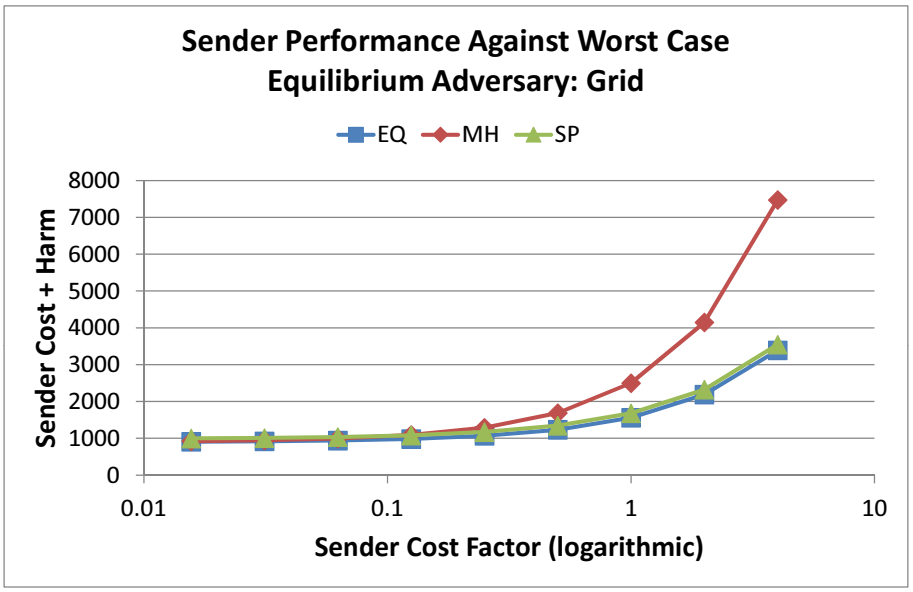


Figure 5.13: Total costs to the sender when playing an equilibrium (EQ), minimum harm (MH), or shortest paths (SP) strategy against an adversary playing a worst case equilibrium strategy on a grid.

MH.

However, there is also the issue of security. When playing MH or SP the sender is playing off-equilibrium. If the adversary knows that the sender is playing off-equilibrium, there is no reason for him to continue to play an equilibrium strategy of his own. Instead, he will play a strategy that will maximize his own payoff against the sender's off-equilibrium play. We consider the adversary playing a worst case best response, that is, a best response that has the worst payoff for the sender. This reflects the case where the adversary can either observe the sender's actual strategy or knows which type of strategy the sender is playing (MH or SP). It also establishes the security of the game of the sender, that is, the worst case outcome against a rational adversary.

The total costs to the sender when playing against an adversary playing worst case best response is shown in Figures 5.16 –5.18. In Figure 5.16 we clearly see that the SP strategy does very poorly compared to EQ and even MH except for high values of γ. The reason for this is that SP is very vulnerable to attack: it chooses a single path through the network, which means the adversary can choose a pure strategy that maximizes the harm.

This is effect still present for the disk and Afghanistan graphs, but it is more difficult to see

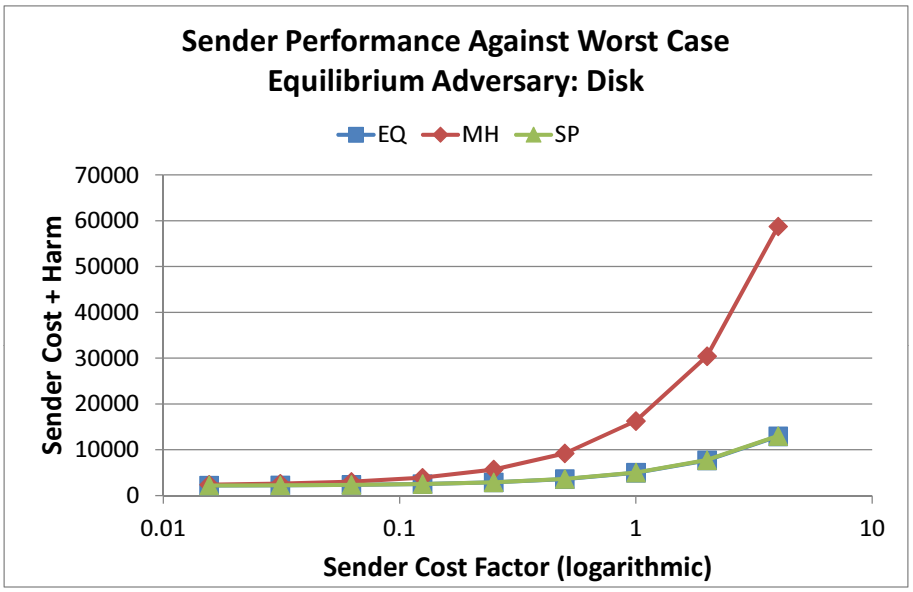


Figure 5.14: Total costs to the sender when playing an equilibrium (EQ), minimum harm (MH), or shortest paths (SP) strategy against an adversary playing a worst case equilibrium strategy on a disk network.

due to the environmental costs dominating the total costs, especially for the MH strategy. This makes the relative contribution from SP's greater vulnerability to attack harder to see. Careful examination of Figure 5.17 shows that there is a significant gap between the curves for EQ and SP, but this cannot be visually seen in Figure 5.18.

A better way to visualize this vulnerability is to consider the relative security gap. The security gap is the difference in payoffs to the sender when the adversary plays a worst case equilibrium strategy and when the adversary plays a worst case best response. The relative security gap then normalizes this to the sender's payoff against the equilibrium adversary. Thus, the relative security gap reflects the sender's vulnerability as the percentage difference in payoff between an equilibrium adversary and a best response adversary.

The relative security gap is plotted in Figures **??** – **??** for the three topologies; a more negative value indicates worse performance. We can see that the SP strategy performs poorly in all three topologies for small values of γ. As γ increases, the gap narrows as the total costs, and hence the denominator, increase. Still, it is very clear from these graphs that the increase in total costs for MH is due entirely to the rising environmental costs as γ increases; MH has a small security gap and hence is not very vulnerable to attack, as is expected.

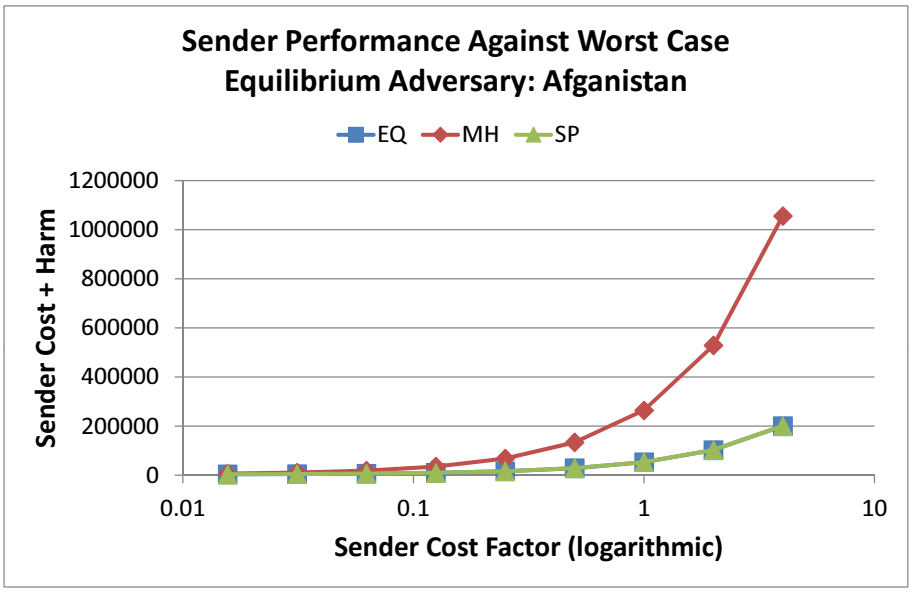


Figure 5.15: Total costs to the sender when playing an equilibrium (EQ), minimum harm (MH), or shortest paths (SP) strategy against an adversary playing a worst case equilibrium strategy on the Afghanistan road network.

From this set of experiments we can conclude that the equilibrium strategies found by LP 10 offers the best of both heuristics: it provides the security of MH and the good performance of SP, trading off the risks of harm and environmental costs in an intelligent way. This is the goal of our work on flow allocation in adversarial environments, and our approach meets that goal.

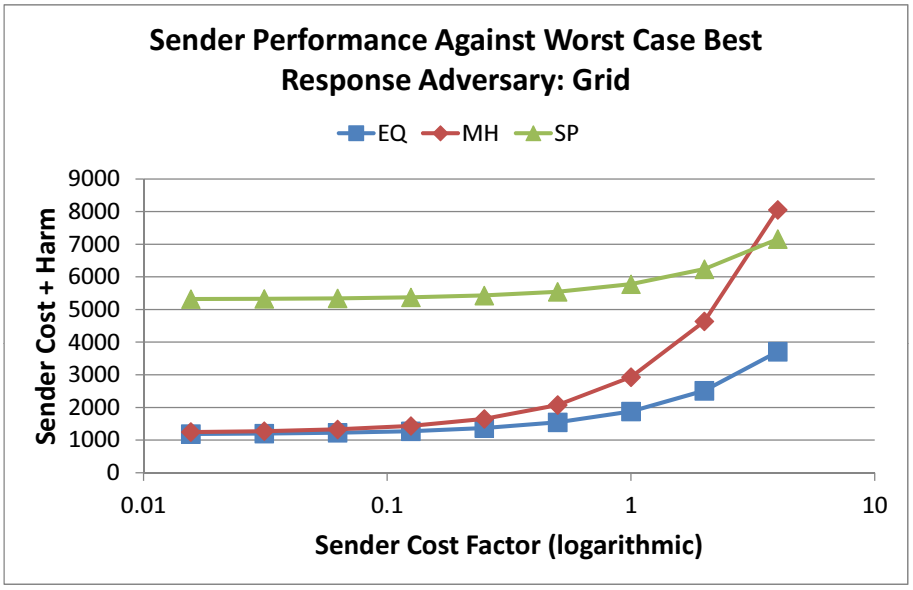


Figure 5.16: Total costs to the sender when playing an equilibrium (EQ), minimum harm (MH), or shortest paths (SP) strategy against an adversary playing a worst case best response strategy on a grid.

5.7 Stackelberg Games

In this section we consider the Stackelberg game in which the sender plays first, committing to a strategy. We show how two commonly used solution concepts, the strong and weak Stackelberg equilibria, are inappropriate for sequential network security games, and provide a polynomial time algorithm for finding a more nuanced equilibrium.

5.7.1 Model

In the previous section we described the simultaneous game where the sender and adversary act without observing each other's actions. However, in many settings this is not the case. For example, convoys in support of persistent military or humanitarian relief missions will operate over extended periods of time and the adversary can observe routes taken over time to build up an estimate of the sender's mixed strategy before choosing which attacks to launch. These types of settings are commonly modeled as *Stackelberg games*, a type of sequential game in which one player (the "leader") moves first, committing to a mixed strategy. The second player (the "follower") can then observe that mixed

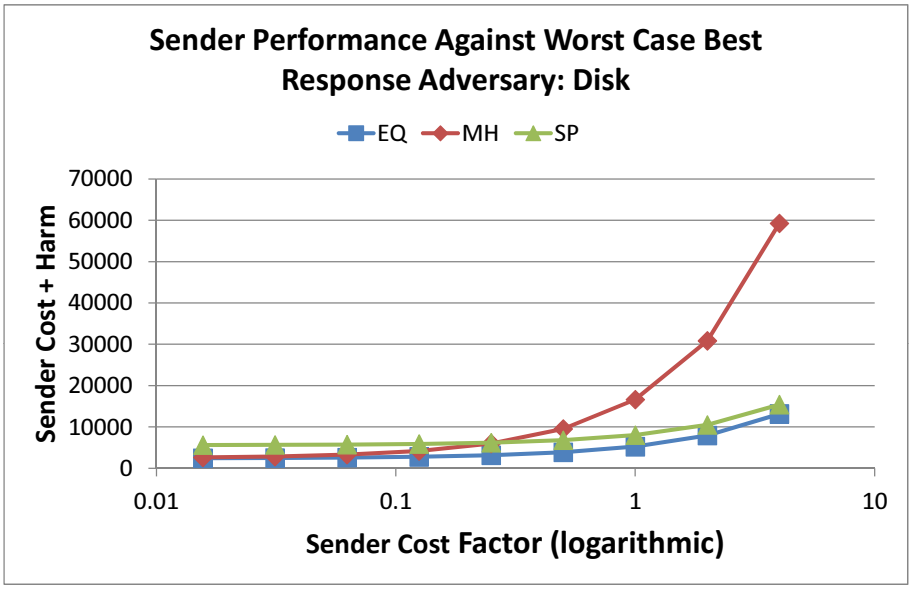


Figure 5.17: Total costs to the sender when playing an equilibrium (EQ), minimum harm (MH), or shortest paths (SP) strategy against an adversary playing a worst case best response strategy on a disk network.

strategy and choose an appropriate response. It is known that in Stackelberg games the leader can sometimes improve his equilibrium payoff (and cannot decrease it, under mild assumptions) compared to his equilibrium payoff in the simultaneous move game [59].

In a two-player Stackelberg game the follower's strategy is a function that maps mixed strategies of the leader to mixed strategies of the follower. In the network flow security game with attack costs, the adversary's strategies are functions $g : \mathcal{F} \to \mathcal{A}$ that map each flow to an adversary mixed strategy. Let $\mathcal{G}$ denote the set of all such functions. A Stackelberg equilibrium is a refinement of subgame perfect Nash equilibrium where (f^*, g^*) are a Stackelberg equilibrium if they are mutual best responses, that is

$$g^*(f^*)Mf^* = \max_{f \in \mathcal{F}} g^*(f)Mf$$
$$g^*(f^*)Mf^* - g^*(f^*)c = \max_{g \in \mathcal{G}} g(f^*)Mf^* - g(f^*)c.$$

both hold, and $g^*(f)$ is a best response to f for all $f \in \mathcal{F}$ (the follower always plays optimally, even off the equilibrium path). Computing a best response function g for the adversary is straightforward: given f, greedily choose up to k attacks that have maximum

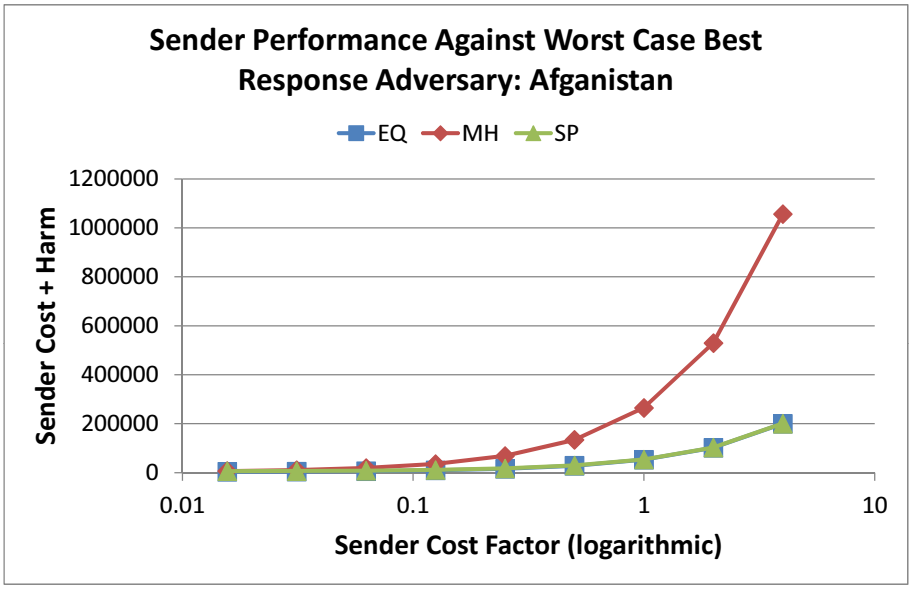


Figure 5.18: Total costs to the sender when playing an equilibrium (EQ), minimum harm (MH), or shortest paths (SP) strategy against an adversary playing a worst case best response strategy on the Afghanistan road network.

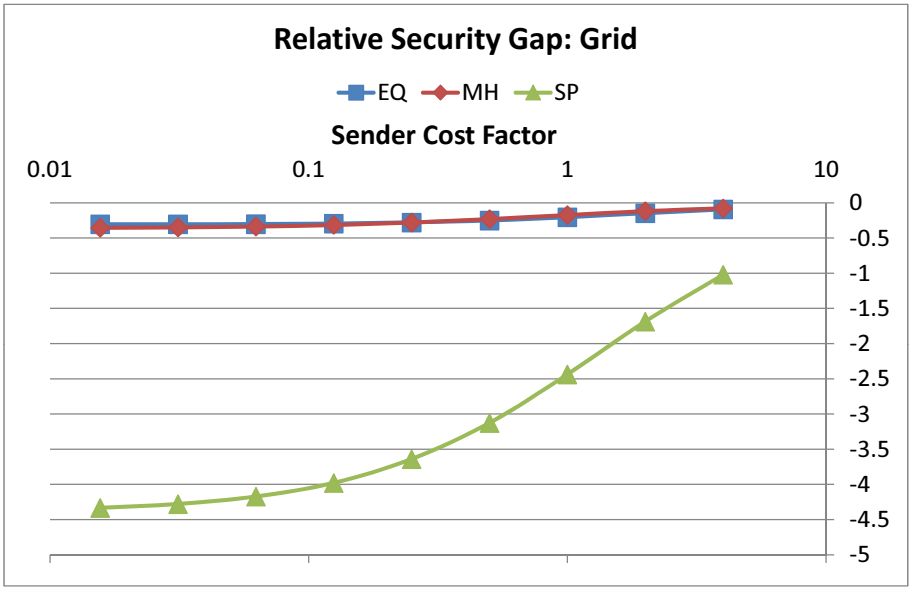


Figure 5.19: Relative security gap when the sender plays an equilibrium (EQ), minimum harm (MH), or shortest paths (SP) strategy on a grid network.

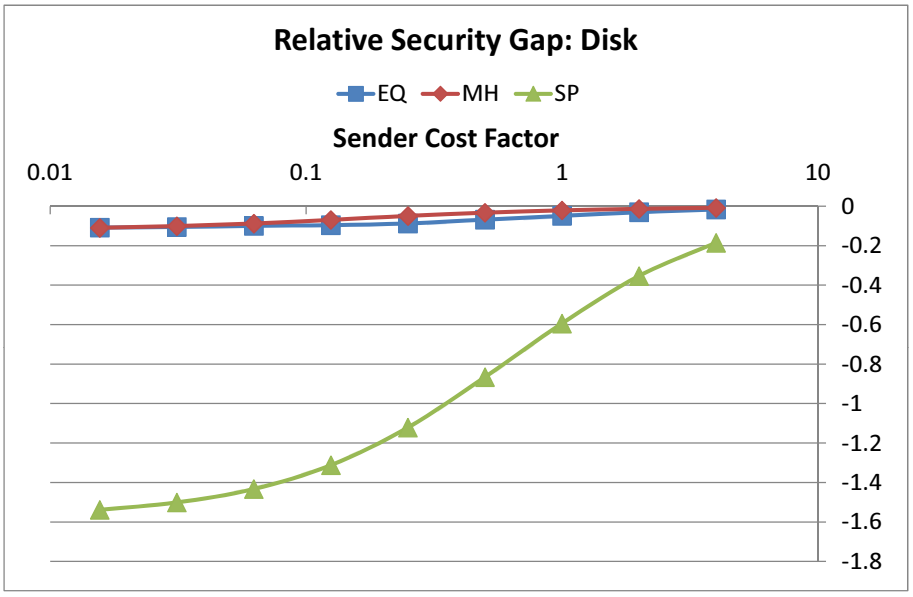


Figure 5.20: Relative security gap when the sender plays an equilibrium (EQ), minimum harm (MH), or shortest paths (SP) strategy on a disk network.

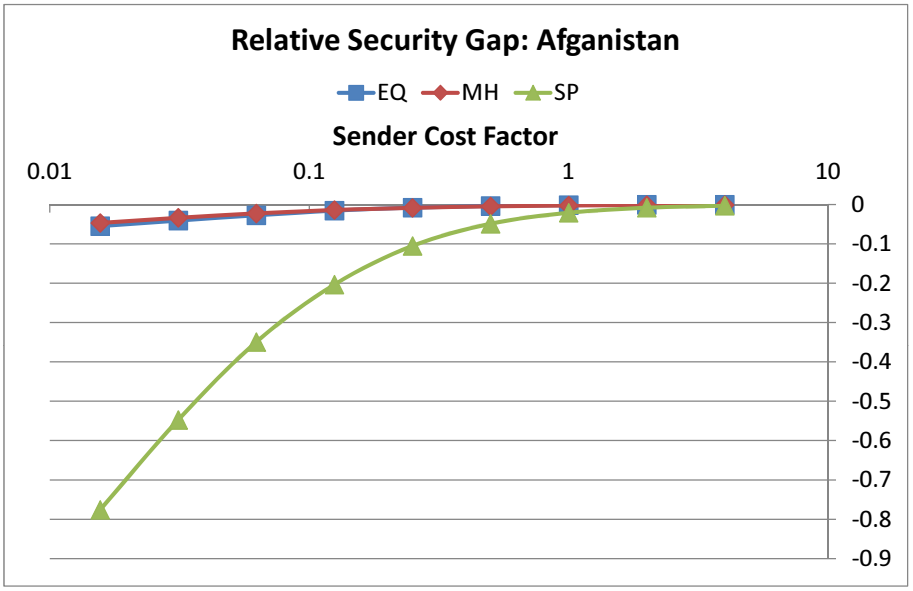


Figure 5.21: Relative security gap when the sender plays an equilibrium (EQ), minimum harm (MH), or shortest paths (SP) strategy on the Afghanistan road network.

payoff to the adversary, excluding any that would contribute negative payoff because the attack cost is too high. Note that there will be multiple best response functions if there is some f for which the set of k attacks yielding highest adversary payoff is not unique, and that these best response functions may yield different payoffs to the sender because of heterogeneous attack costs. Thus there may be multiple Stackelberg equilibria that have the same sender strategy but different sender payoffs.

Traditionally two kinds of Stackelberg equilibrium are distinguished: strong Stackelberg equilibrium (SSE), where the follower's best response function always maps to a strategy that maximizes the leader's payoff; and weak Stackelberg equilibrium (WSE), where the follower's best response function always maps to a strategy that minimizes the leader's payoff [34]. The pessimistic WSE is the more natural solution concept for security applications, which tend to focus on worst case behavior. Despite this, SSE, which assumes that the malicious adversary breaks ties in the leader's favor, has been considered more often in the literature for two technical reasons: (1) a SSE is guaranteed to exist in every Stackelberg game, while a WSE may not; and (2) it is often claimed that the leader can *induce* the adversary to play the desired best-case strategy by deviating by an arbitrarily small amount from the equilibrium in order to break the adversary's indifference [59]. We will show that both of these arguments are inappropriate for the network security game, but first illustrate several important concepts by example.

Recall the example in Figure 5.1 with $c_1 = 102$. The adversary is indifferent when $f_1 = 103/105$, prefers the top path when it is $f_1 > 103/105$, and prefers the bottom path when $f_1 < 103/105$. Thus all best response functions $g_1 : [0,1] \to [0,1]$ mapping f_1 to the probability of attacking the top path must satisfy $g_1(f_1) = 0$ when $f_1 < 103/105$ and $g_1(f_1) = 1$ when $f_1 > 103/105$, and any value $g_1(f_1) \in [0,1]$ is acceptable for $f_1 = 103/105$.

In the unique simultaneous Nash equilibrium, $p_1 = 3/105$, so that the sender was indifferent between the top and bottom paths but sent $f_1 = 103/105$ flow on the top path and $f_2 = 2/105$ flow on the bottom path, suffering harm on both paths. It follows that $f_1 = 103/105$ is a best response to the adversary's best response function g_1^{NE} with $g_1^{NE}(103/105) = 3/105$. Thus the simultaneous Nash equilibrium naturally gives rise to a Stackelberg equilibrium strategy, the same payoff to the sender as in the Nash equilibrium, $-306/105$. In the SSE the adversary attacks the bottom path ($g_1^{SSE}(103/105) = 0$), resulting in a much higher payoff to the sender, $-6/105$. It is easy to see that there are no other Stackelberg equilibria for this game. For example, there is no WSE because if the adversary played the worst-case best response with $g_1^{worst}(103/105) = 1$, then the the sender would have incentive to deviate by decreasing f_1.

The sender's strategy is the same in both of these equilibria which means that his payoff

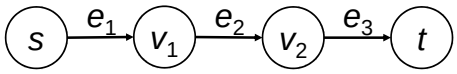


Figure 5.22: A network topology in which the sender cannot induce a strong Stackelberg equilibrium.

ultimately depends on the choice of the indifferent adversary. However, note that the sender can deviate slightly from his equilibrium strategy by playing $f_1 = 103/105 - \varepsilon$ for some small $\varepsilon > 0$, in order to incentivize the adversary to attack the bottom path. By doing this the sender will receive a payoff of $6/105 + 3\varepsilon$ instead of the $6/105$ that he would earn in the SSE, but as ε is made arbitrarily small his strategy converges to the SSE strategy.

It is not always possible to induce the SSE by deviating from an equilibrium strategy. Consider the network in Figure 5.22, and assume that A contains two attacks, one that affects e_1 and one that affects e_2, with harm matrix

$$M = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \end{bmatrix}$$

and costs $c_1 = 3$ and $c_2 = 1$. The sender has no choice as his only pure strategy is to send the full flow on the single path from s to t. At the same time, the adversary is indifferent to the choice of attack as they both yield him a payoff of 2 and so might choose either of them.

5.7.2 Inducing Locally Optimal Equilibria

Given that the WSE may not exist and the SSE might not be attainable, we solve the problem of how the sender can deviate from a Stackelberg equilibrium strategy f to induce a Stackelberg equilibrium (f, g) that yields him maximum payoff. We call this equilibrium a locally optimal induceable Stackelberg equilibrium (loptISE). It is locally optimal because the value of the Stackelberg equilibrium that is induced depends on the starting equilibrium strategy f. The starting strategy that we use is one that arises naturally from simultaneous game Nash equilibrium strategy as solved for using LP 9, which can be shown to always be a Stackelberg equilibrium:

Lemma 6. *A strategy profile (f, p) is a Nash equilibrium for the network flow security game with attack costs if and only if (f, g) is a Stackelberg equilibrium for the Stackelberg network flow security game with attack costs for a best response function g with $g(f) = p$.*

Algorithm 5 Computing deviation to find optimal inducible Stackelberg equilibrium

Find Nash equilibrium flow f using LP9
Set A' to be the set of minimum adversary payoff candidate attacks.
Set $k' \leq k$ to be the number of candidate attacks that must be chosen from A'.
if $|A'| \leq k'$ **then**
    Return.
Add dummy source s_0 to G. Set $I' \leftarrow \emptyset$. Set f^{ε} to be the empty flow.
while $|I| < k'$ **do**
    Set $F \leftarrow \emptyset$.
    for all $a \in A'$ **do**
        Solve $(f^a, H, H') \leftarrow$ LP13
        if $H - H' \geq 0$ **then**
            $F \leftarrow F \cup \{f^a\}$
    Set $Y \leftarrow \{a | f^a \in F$ with minimum $\text{row}_a[M] f^a\}$.
    Set $Z \leftarrow \{a | \exists a' \in Y$ s.t. $\text{row}_a[M] f^a = \text{row}_{a'}[M] f^{a'}\}$
    Set $A' \leftarrow A' \setminus Z$ and $I \leftarrow I \cup Z$.
    Set $f^{\varepsilon} \leftarrow f^{\varepsilon} + \sum_{a \in Y} f^a$
Set $f \leftarrow f + \varepsilon f^{\varepsilon}$
for all $s \in S$ **do**
    Normalize outgoing flow.

Proof. Suppose (f, p) is a Nash equilibrium and construct g as a best response function with $g(f) = p$. By definition of Nash equilibrium f and $g(f)$ are are mutual best responses because and so (f, g) is a Stackelberg equilibrium. Suppose (f, g) are a Stackelberg equilibrium. Then by definition of Stackelberg equilibrium f and $g(f) = p$ are mutual best responses and hence (f, p) is a Nash equilibrium. □

Algorithm 5 computes a deviation from equilibrium strategy that the sender can use to induce a loptISE. We will first sketch the high level approach before delving into the details. The algorithm starts from a Nash equilibrium flow f that will also be a Stackelberg equilibrium strategy according to Lemma 6. It then computes the set A' of *candidate attacks* for the adversary. These are the attacks that might be chosen as part of a best response to f. The sender will try to incentivize the adversary to choose certain of these candidate attacks by adding small amounts of flow to certain paths. Intuitively this approach exploits what we observed in the example: parallel paths allow the sender freedom to deviate and bias the adversary; a sequential topology does not permit that flexibility. However, the process is not as obvious when dealing with general attacks, each of which may affect

an arbitrary set of links with heterogeneous penalties. Instead of choosing a simple path, the sender tries to find a flow for each candidate attack that will cause the adversary to prefer to play that attack. If the attacks are "in parallel" this will be possible, if the attacks are "in sequence" it will not be. When presented with multiple options, the sender will choose the one that causes the least harm (i.e., increases his payoff the most). The process repeats until the sender has guided all of the adversary's attacks, or he has guided all the attacks that are possible. The deviation flows (which may be made arbitrarily small) are then superimposed on the original flow to generate the desired deviation.

Computing the candidate attacks is straightforward. Given f, we compute the payoff that the adversary would receive for each attack $a \in A$ as $\rho_a = \text{row}_a[M]f$. If there are k or fewer attacks with $\rho_a \geq 0$, then those are all candidate attacks. If there are more than k such attacks, we compute A' as the set of attacks with the highest k values of ρ_a (with repetition). For example, if the set of ρ_a is $\{10, 10, 8, 7, 7, 7, 0, -2\}$ and $k = 4$, then $A' = \{a \in A | r_a \geq 7\}$, giving us 6 candidate attacks. The adversary's best response will always choose the attacks with ρ_a strictly greater than the minimum, so we need only consider A' to be those with minimum ρ_a values. In the previous example, that means that the best response always plays the two attacks with $\rho_a = 10$ and the one attack with $\rho_a = 8$, so we are left to choose $k' = k - 3 = 1$ candidate attacks from among the three remaining with $\rho_a = 7$.

A dummy source node s_0 is added to G and connected to each source in $s \in S$, allowing deviant flows from any source. I, the set of induced attacks, is initialized as empty. The overall deviant flow f^{ε} is initially empty.

The algorithm then iterates up to k' times in the loop starting at line 7. On each iteration it attempts to greedily induce the adversary to choose attacks that maximally increase the sender's payoff. F is the set of best deviant flows, computed for each candidate attack in the loop starting at line 9. The best deviant flow for a candidate attack a is computed by LP13. This LP finds a flow that causes as great as possible an increase in harm for attack a compared to any other candidate attack. For a deviation flow f^a, the adversary's best response is an attack with maximum increase in harm (attack cost does not matter as the adversary is already indifferent between candidate attacks due to the Nash equilibrium sender strategy), so if the objective value is non-negative, the adversary can be induced to play a (and possibly other candidate attacks as well). If the objective value is negative, the adversary cannot yet be induced to play a in preference to other candidate attacks.

Of the attacks that can be induced on this iteration, the sender chooses those that cause minimum increase in harm. These may not be unique, so Y is the set of all such candidate attacks with minimum increase in harm that the adversary can be induced to attack on this iteration. The deviation flows that are used to induce these attacks may also induce other

LP 13 Stackleberg deviating flow for a.

Input: G, M, A', a
Output: f^a, H, H'

$$\underset{f,H,H'}{\text{Maximize}}\ H - H' \tag{5.75}$$

subject to:

$$H' \geq \text{row}_a[M] f^a \tag{5.76}$$

$$H = \text{row}_{a'}[M] f^{a'} \qquad \forall a' \in A' \setminus \{a\} \tag{5.77}$$

$$\sum_{(v,u)\in E} f_{vu} = \sum_{(u,v)\in E} f_{uv} \qquad \forall v \in V \setminus \{s_0, t\} \tag{5.78}$$

$$\sum_{(s_0,u)\in E} f_{s_0 u} = 1 \tag{5.79}$$

$$f_{uv} \geq 0 \qquad \forall (u,v) \in E \tag{5.80}$$

$$\lambda_a \geq 0 \qquad \forall a \in A \tag{5.81}$$

attacks (which have the same increase in harm), so the set Z contains all the attacks that will be induced in the current iteration. These are removed from the candidate attacks and added to the induced attacks in line 15, and the deviation flows for this iteration are superimposed on the total deviation flow f^ε before starting a new iteration.

The loop terminates when the requisite number of attacks have been induced. The case when no more candidate attacks can be induced is handled by there being a single flow in F which is trivially deviated from itself. For performance considerations this possibility can be checked separately. It is also not possible for A' to become empty prior to the termination of the loop. Recall also that at the beginning of iteration, $|A'| > k'$ (lines 4 – 5) and on every iteration the same number of attacks are added to I as are removed from A'. Thus, $|I| \geq k'$ no later than the iteration when $A' = \emptyset$.

In line 17 the deviation flow is scaled and superimposed on the equilibrium flow, and in line 19 the amount of flow (which increased due to the addition of the deviation flow) is normalized at each source node so that the total amount of flow is maintained with the addition of the deviation.

Theorem 5. *Algorithm 5 runs in time polynomial in the size of G and A.*

Proof. Each line in the algorithm can clearly be executed in polynomial time. The for loops in lines 9 – 12 and lines 18 – 19 iterate at most $\Theta(|A|)$ and $\Theta(n)$ time, respectively.

In each iteration of the main loop from lines 7 – 16 at least 1 attack is added to $|I|$ and therefore the loop cannot iterate more than $|I| = \Theta(|A|)$ times. $\square$

Chapter 6

Related Work

6.1 Flow Allocation

The issue of transmitting data to a central base station has received considerable attention in ad hoc and especially sensor networks [29, 28, 39, 1]. The primary concern for many algorithms operating in these kinds of networks is power usage, and to a lesser extent, scalability and simplicity. This has led to many algorithms that give rise to tree or tree-like topologies, such as through hierarchical or cluster-based routing [28, 39], data aggregation [1, 19], or topology control [49]. By focusing on tree topologies, we can address partial centralization problems without committing to a specific network algorithm.

A number of well-known network design problems are also related to the flow allocation problems we consider in this paper. Given an undirected graph, a root node, a set of source nodes and their demands, and a uniform edge capacity, the capacitated minimum Steiner tree problem (CMStT) is to find the minimum cost Steiner tree in which all source nodes can route their flows to the root without violating the edge capacity constraints. A $(\gamma\rho_{ST} + 2)$-approximation was given in [33], where γ is the Steiner ratio and ρ_{ST} is the best achievable approximation algorithm for the Steiner tree problem. There are several key differences from the flow allocation problem. First, CMStT seeks to minimize edge costs for routing all flows, while our objective is to maximize the number of source nodes whose full flows can routed to the sink. Second, CMstT assumes uniform edge capacities, while in general we do not.

In the unsplittable flow problem (UFP) you are given a graph and source-sink pairs along with a given demand, and the problem is to find a single path from each source node to its sink along which its flow can be routed. There are three variants, all NP-hard: maximizing

the amount of flow transmitted, partitioning the sources into the minimum rounds such that all sources within a round can transmit their flows simultaneously, and minimizing the maximum congestion along an edge in the network. Several approximation algorithms for these problems have been devised [36, 11]. However, none of these consider the problem where groups of source nodes must transmit to a common sink. Our proof of Theorem 2 follows a proof from Guruswami, et al. [26] of the hardness of approximating the edge disjoint paths problem (EDP) of finding a path from each source to its paired destination, without any of the paths sharing an edge in common.

6.2 Network Augmentation

Network augmentation combines three fundamental problems: supplemental node deployment, the assignment of groups to sinks, and flow allocation on the edges. There has been considerable research related to each of these problems individually, but never for all three of them simultaneously. The network augmentation problem is very similar to facility location problems that have been extensively studied in the operations research community [40, 67, 5, 45]. Given a number of facilities and a set of sites with quantities of goods demanded at each site, the facility location problem is to place the facilities in order to satisfy the demand of the sites for minimum transportation costs of goods from facilities to sites. Variants to the facility location problem include capacitated problems [14, 45] in which there are capacity constraints on facilities or on transportation links, as well as multiple commodity problems [54] where different facilities can provide different types of goods. One major difference between facility location problems and the network augmentation problem is that they assume a direct link between facility and site, while communication from group member to sink can be through a multihop network. Another is that the flows from groups of source nodes must be transmitted to the same sink.

One area of networking research focuses on how to provide access between two initially disconnected networks through the addition of additional hubs, routers, and bridges. The problems addressed in that literature are similar to the network augmentation problem. One approach is to partition the network into multiple local access networks (LANs) and a backbone network [2, 56]. Each LAN has one node that is designated the access point and is also part of the backbone network. Traffic between nodes in different LANs must first be routed to the access point for the originating LAN, conveyed across the backbone network, and then routed across the destination LAN. Links are considered costly and capacitated. The local access network design problem is to design the local access networks by purchasing LAN edges between nodes so that the total cost is minimized and a known

amount of traffic can be routed to the access points, which has similarities to both network augmentation and flow allocation. The access network design problem is NP-hard, but it is known that there exist optimal solutions in which the LANs take the form of trees with the access points as roots [2]. Linear programming formulations have been used to approximately solve the access network design problem in [2] and more recently in [56]. Unlike the network augmentation problem considered here, however, all inter-LAN traffic must be conveyed through the backbone network. In addition, the communication pattern is single-source-single-destination, instead of the centralizing groups we consider here.

The formation of two-tiered communication networks has also been a focus in mobile ad hoc networking [4, 23, 12]. In these networks, there is no pre-existing "backbone network" and the problem is to dynamically create such a network from the underlying ad hoc network. This is accomplished by partitioning the nodes into clusters, and selecting a clusterhead node in each cluster to act as the access node for the backbone network. As result, all intra-cluster traffic is conducted in multi-hop through the ad hoc network, but inter-cluster traffic must pass through the clusterhead, which then relays it to other clusterheads. The primary focus of research has been in developing techniques of cluster formation and clusterhead selection, such as highest ID [4], highest degree [23], node weight [6], and weighted clustering [12]. These techniques primarily focus on metrics such as cluster stability and power conservation, and ignore communication costs, capacities, and specific communication requirements. In addition, they assume that the nodes (including the clusterheads) move exogenously, while we actively position of the supplemental nodes specifically to meet the demands of the network.

Virtual private network (VPN) provisioning [25] is one of the few areas that consider a group communication pattern. In VPN provisioning, groups of nodes within a network wish to form a subnetwork by reserving bandwidth from the underlying network. Given bounds on the communication demands of the nodes that wish to form the VPN, the VPN provisioning problem is to reserve bandwidth so that any traffic pattern respecting the given bounds can be feasibly routed. Polynomial-time optimal and approximation algorithms were found for some problems, but the capacitated version of the problem is NP-hard [25]. The "group communication" in VPNs differ from that in this thesis because VPN member communicate with each other, while centralizing group members transmit data to the centralization point. Also, while VPN provisioning allocates bandwidth for a group-oriented communication pattern, it does not address the issue of supplementing the network through additional nodes.

6.3 Security in Adversarial Environments

Choosing paths through hostile environments have been studied in operations research [66, 30], robotics [27, 7], and multiagent systems [62, 32]. Many of these have also taken the perspective of the player who selects nodes or edges in the network to impair the other player who chooses paths through the network. The study of network interdiction [66, 30] looks at problems where an interdictor chooses edges or nodes to damage or destroy destroy ("interdict") in order to impair the ability of an enemy moving through the network, for example for by forcing it to take longer paths [30]. An early study of single source, single sink zero-sum games where the interdictor interdicts a single edge found that the equilibrium strategy is to only interdict edges in the minimum cut [66]. Similar results were found in network routing settings [10], and more recently in games where multiple edges can be interdicted[62, 32]. In evader-pursuer games [27, 7], both players move through the network. In path disruption games [3] multiple cooperative agents work together to interdict an adversary, in contrast to our setting where both sides are assumed to be monolithic players.

In most of these related problems, the payoff depends on the probability that at least one attack occurs on a pathway; multiple attacks on the same pathway either are not possible or incur no additional penalty. This models situations like placing checkpoints to intercept the sender; once caught, the sender cannot be caught again. In contrast, in our problem the same pathway may be subject to multiple attacks or a single attack may affect multiple edges on the same pathway, resulting in additional harm. This is useful for settings where the sender continues after an attack, as when convoys fight their way through ambushes or robots clear obstacles. Games with similar payoffs have been solved in the context of Markov Decision Processes using oracle algorithms [43]. However, these approaches have assumed only zero-sum games.

A class of security games between two players, an attacker and a defender, have recently been proposed and studied in a variety of contexts [52, 64, 34, 69, 62, 32, 63]. The attacker chooses targets to attack from a known set of targets. The defender attempts to foil attacks by assigning defensive resources from a known set of defensive resources to a known set of schedules, where there is a known subset of schedules that each resource can be assigned to. Each schedule protects or *covers* a subset of the targets. For each target that is attacked, the attacker and defender receive payoffs depending on whether the target is covered or not, and when multiple targets are attacked the payoff is the sum of the payoffs for each target. A key characteristic of these security games is that the defender receives a higher payoff when a target is covered than when it is uncovered and the attacker receives a higher payoff when a target is uncovered than when it is covered. Crucially to comparison with

this thesis, the payoffs for a target is the same if is attacked one or more times, or if it is covered by one or more resources.

Yin et al. [69] studied the equilibrium properties of these types of security games, examining the relationships between minimax strategies, Nash equilibria, and strong Stackelberg equilibria for Stackelberg games where the defender is the leader and the attacker is the follower. They proved that when the attacker can only attack a single target, the defender's minimax and Nash equilibrium strategies are identical, and that all Nash equilibria are interchangeable. Furthermore, given the additional assumption that any subset of a defender's schedule is itself a schedule, they showed that the defender's SSE strategies are also Nash equilibrium strategies.

There are many similarities between these security games and the games considered in this thesis. However, careful examination reveals that the games in this thesis do not satisfy the assumptions of the security games as defined by Yin et al. and others. The key difference is in the way payoffs are computed in the two models. In Yin et al., payoffs are computed based on the binary conditions of whether the target is attacked or not attacked, and covered or uncovered. In this thesis, the central component of the payoff, the harm, depends on the specific attack and path chosen for the flow, with harm summed over multiple attacks and paths.

We can demonstrate this fundamental difference with a simple example of a zero-sum game. Because the security games of Yin et al. are played over finite strategy spaces, a natural comparison is with the zero-sum path game (ZS-PG) of this thesis, with network flows then understood to compactly represent mixed strategies over the combinations of paths. Consider the graph with a single source node, a single sink node, and three paths from the source to the sink, as shown in Figure 6.1. The sender has three pure strategies: π_1, the path through v_1; π_2, the path through v_2, and π_3, the path through the v_3.

Suppose that the adversary has three attacks, $A = \{a_1, a_2, a_3\}$, and that he can play a single attack at a time (i.e., $k = 1$) so that he also has three pure strategies. Each attack affects two edges, causing harm according to the following harm matrix:

$$H = \begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 & 0 \\ 1 & 0 & 2 & 0 & 0 & 0 \end{bmatrix} \tag{6.1}$$

Note that the effect on the harm is unequal between the two edges affected by each attack. For example, when the adversary plays attack a_1, the sender suffers 2 units of harm if he plays π_1 (which includes e_1) and only 1 unit of harm if he plays π_2 (which includes e_2).

We can represent the payoffs explicitly using the normal form, where the sender is the row

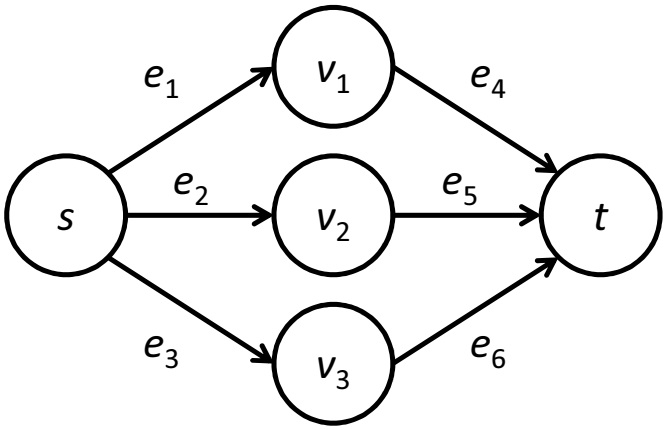


Figure 6.1: Graph for counter-example of NFG as a security game.

player and the adversary is the column player. This gives us the following payoff table:

$$\begin{bmatrix} -2,2 & 0,0 & -1,1 \\ -1,1 & -2,2 & 0,0 \\ 0,0 & -1,1 & -2,2 \end{bmatrix} \tag{6.2}$$

These payoffs do not satisfy the structure of a security game as defined by Yin et al. because it cannot be defined in terms of covered and uncovered targets that are either attacked or not. For example, when the adversary plays a_1, the sender can receive one of three possible payoffs: -2, -1, or 0. That cannot be represented as an attacked target being either covered or uncovered (which would lead to only two possible payoffs). Nor does any pure strategy dominate any others, allowing us to effectively reduce the strategy spaces to ones where we can represent this game as a security game. This example shows that the games studied in this thesis, even the simplest zero-sum, single-source, single-sink games, are not a subset of the security games previously addressed in the literature, and thus the theoretical results that have been established in that literature is not directly applicable to our games.

Instead we can look to the concept of strategically zero-sum games defined by Moulin and Vial [44]. These are a generalization of zero-sum games that share many properties of zero-sum games, such as equilibria strategies being minimax strategies and equilibria being interchangeable. Lemma 2 of this thesis establishes that the non-zero-sum network flow game is strategically equivalent (as defined by Moulin and Vial) to the zero-sum game with payoffs defined by $U(f, q)$ in Equation (5.1), thus meeting the definition of a strategically zero-sum game.

Stackelberg games [50, 62] have recently been used to model security games where patterns of behavior may be observed and learned by the adversary, as opposed to more traditional simultaneous games[10, 42]. Stackelberg games generally allow the leader to find equilibrium strategies with higher payoff than in a simultaneous game, but only in non-zero sum games [69, 59]. Computing the optimal strategies to commit to is solvable in polynomial time in the normal form game [13], but this is not practical in our games which have exponential-sized strategy spaces. The traditional solution concept considered in all of these is the strong Stackelberg equilibrium, which is questionable for the worst-case reasoning common in security settings and is not appropriate for the network security games we consider.

Chapter 7

Conclusion and Future Work

This thesis addressed the problem of flow allocation to support multiagent task execution. Performing tasks requires inputs to be brought to the task execution location: performing a physical task requires agents and resources to move through the physical environment to the location of the task, while performing a computational task requires input data to be transmitted through the communication network to a computational platform where the computation can occur. This thesis modeled the movement of these inputs as network flows, formulated and analyzed problems in allocating flows in several kinds of environments, and presented algorithms for solving these problems. This chapter summarizes the major results of this thesis and discusses some lines of future work.

7.1 Summary

This thesis addressed the problem of flow allocation in three environmental contexts. Chapter 2 formulated the flow allocation problem for known, capacitated environments. This approach was extended in Chapter 3 to capacitated environments that could be augmented by the agents to add supplemental nodes and edges. Chapters 4 and 5 addressed flow allocation in costly environments where costs were affected by an adversary.

7.1.1 Capacitated Environments

Chapter 2 formulated the flow allocation problem using a directed, capacitated graph for the environment, groups of source nodes for agent subteams, and sink nodes for task ex-

ecution locations. The core flow allocation problem is to assign sinks to groups and find flows from the source nodes to the sink assigned to their groups.

This thesis makes three main contributions:

1. Maximizing the number of groups transmitting their full amount of flow to their assigned sink is formulated as the Maximum Satisfied Group (MaxSG) problem. This corresponds to maximizing the number of tasks that can be executed. When flows can be divided on multiple paths, as arises with data streams in communication networks or some kinds of resources in physical networks, MaxSG on general graphs is strongly NP-hard and so there are no pseudo-polynomial time algorithms known for solving it optimally.
2. A pseudo-polynomial time algorithm for solving MaxSG with divisible flows for graphs with tree topologies, as arise in some kinds of communication networks,
3. Proof that when flows cannot be divided among multiple paths, for example with embodied agents that must take a single path from the source to the sink, flow allocation is not only NP-hard to solve optimally, but NP-hard to approximate better than satisfying the requirements of a single group.

7.1.2 Network Augmentation

Chapter 3 considered flow allocation in capacitated graphs where the agents could augment the network by adding supplemental nodes and edges. This models communication networks that can be enhanced through the addition of additional dedicated relay nodes, or physical environments where new movements can be enabled through the removal of obstructions. The potential network represented the locations where supplemental nodes could be deployed and the edges that would result if they were. Two main problems were addressed: maximizing the number of satisfied groups with a fixed number of supplemental nodes (MaxSG-NA) and minimizing the number of supplemental nodes while satisfying all groups (MinDep). Solving either of these required finding deployments of supplemental nodes to potential locations (network augmentation), as well as finding sink assignments and flows from the source nodes to the assigned sinks (flow allocation).

This thesis makes three main contributions:

1. Augmenting a network to provide connectivity to source node groups is NP-hard.
2. Mixed integer linear programs for solving the NP-hard problems MaxSG-NA and MinDep optimally.
3. Heuristics for MinDep and MaxSG-NA that iterate through the groups and extended

the supplemental node deployments and sink assignments as each group was considered. These heuristics were empirically shown to dramatically decrease running time at the expense of a modest decrease in solution quality when compared to the optimal algorithms.

7.1.3 Adversarial Environments

Chapters 4 and 5 considered flow allocation in costly graphs where the costs were partially chosen by an adversary. In this setting divisible flows can directly represent the movement of inputs that can be divided among multiple paths (like communication traffic), or may represent probability distributions over the movements of indivisible inputs (like robots). We modeled this setting as a two-player game between a sender (representing the agent team) who chose sink assignments and flows, and an adversary who choose multiple attacks that imposed flow costs on edges. The main problem addressed was to compute equilibrium strategies for the sender.

This thesis makes three main contributions:

1. A polynomially-sized linear program to find equilibrium strategies in the zero-sum setting where the payoffs for the sender and the adversary were directly opposed.
2. A proof that a similar approach can be used in a class of non-zero-sum games where the players unilaterally incur costs based on their chosen strategies, and a linear program for the generalized problem with multiple sinks and multiple groups with these kinds of payoffs.
3. An example that existing solution concepts are inadequate for settings where the sender must commit to a strategy that is then observed by the adversary, and a new equilibrium refinement, the optimal inducible Stackelberg equilibrium, to address this shortcoming, and an algorithm for computing a sender strategy for it.

7.2 Future Work

In this section we describe several areas of future work for flow allocation.

7.2.1 Generalized Task Structures

This thesis considered relatively simple task structures: each task required inputs from a known set of agents and sets of agents performed a single task. In many multiagent applications the task structures are considerably more complex. Computational tasks often exhibit hierarchical structure, with the outputs of computations at lower levels being used as inputs to computations at a higher level. For example, hierarchical team plans impose a hierarchy on plan monitoring tasks. The plan monitoring task for a team of agents executing a team plan does not need to know the exact status of each agent in the team. Instead, it can use status information from the plan monitors for each subteam of agents performing a subplan. In sensor networks, the output of a computational task like object recognition can then be used as input to another computational task, like threat assessment.

With physical tasks, teams of agents may have to perform multiple tasks. Even if physical tasks require the full attention of the agents performing them, precluding the agents from carrying out multiple tasks simultaneously. However, the agents may still perform multiple tasks over time as they complete a series of tasks, one at a time. For example, a team of firefighters may work to put out a single burning building, but once that fire is extinguished, they are able to move to the another burning building.

A direction for future research is to extend the flow allocation approach taken in this thesis to these kinds of cases. One way could be to allow sink nodes that receive flow to act as source nodes for new flows. For computational tasks, this new flow would represent the output of the computation, and need not be equal to the amount of flow (i.e., input data) that was consumed by the sink. For physical tasks, a new flow would need to be created for each source node that transmitted to the sink, to represent the agent that moved to the task location.

7.2.2 Dynamic Teams, Tasks, and Environments

Dynamism is a major challenge in many multiagent applications. All aspects of a multiagent system may change over time: the agents (or their organization into subteams), the tasks, and the environment. In the context of flow allocation, this means that groups of source nodes, the set of sink nodes, and the graph itself may change over time.

A crude way to deal with dynamism is to re-run the algorithms when things change, but this is often unsatisfactory for many reasons. It can be computationally impractical to repeatedly re-run the algorithms, especially when the changes may not result in a change in the solution, as when the communication network changes slightly due to the movement

of the agents. When the solution does change, small changes in the inputs may result in very large changes to the outputs, which impose hidden costs, as when a deployment of supplemental nodes changes dramatically. Finally, such a scheme is inherently reactive and myopic, which can decrease performance over time.

It may be possible to address these shortcomings by adapting the algorithms in this thesis. Further research may determine what kinds of changes require a solution (or part of a solution) to be recomputed. It could also be possible to mitigate the costs of of changes in the solution, perhaps by explicitly representing those costs in the optimization. A third possibility would be to make the algorithms more proactive, for example by using stochastic programming [15, 9] to reason about the effects of dynamism provided probabilistic information is available.

7.2.3 Partially Distributed Algorithms

The algorithms in this thesis are centralized approaches that rely on full knowledge of the environment. A natural next step is to develop distributed, parallel algorithms to compute flow allocations that do not require full knowledge of the environment at any single location. This could result in more scalable algorithms that do not require the full graph of the environment to be transmitted to, stored at, or computed over at any single location. It could improve running time through parallel computation and reduced input sizes. Although some degree of distribution is desirable, it is not necessary for approaches to be fully distributed. In many distributed multiagent coordination algorithms (for example distributed constraint optimization) it is beneficial to use some amount of centralization in order to reduce communication and redundant computation.

Many multiagent applications feature strong locality. Agents are much more likely to have relatively good knowledge of their local physical and communication environments than they are to know the full environment. Agents are also often located near each other in the environment, and nearby task execution locations are more preferable to more distant ones, all other things being equal. Because of this, it may be possible to compute high quality local solutions based primarily on local knowledge, with a need to communicate global knowledge (in the form of agents, tasks, or topological information) only in a small number of cases. The key will be to efficiently exploit the locality while also being able to handle the rare but often (and often crucial) exceptions.

Bibliography

[1] Kemal Akkaya, Murat Demirbas, and R. Savas Aygun. The impact of data aggregation on the performance of wireless sensor networks. *Wireless Communications and Mobile Computing*, 8:171 – 193, 2008. 2.2, 6.1

[2] Matthew Andrews and Lisa Zhang. The access network design problem. *Foundations of Computer Science, 1998. Proceedings.39th Annual Symposium on*, pages 40–49, 8-11 Nov 1998. 6.2

[3] Yoram Bachrach and Ely Porat. Path disruption games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 1123–1130, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems. 6.3

[4] D. Baker and A. Ephremides. The architectural organization of a mobile radio network via a distributed algorithm. *Communications, IEEE Transactions on*, 29(11):1694–1701, Nov 1981. 6.2

[5] Francisco Barahona and Fabian A. Chudak. *Solving Large Uncapacitated Facility Location Problem*. Kluwer Academic Publishers, 1999. 6.2

[6] Stefano Basagni. Distributed clustering for ad hoc networks. In *ISPAN '99: Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '99)*, page 310, Washington, DC, USA, 1999. IEEE Computer Society. 6.2

[7] N. Basilico, N. Gatti, and F. Amigoni. Leader follower strategies for robotic patrolling in environments with arbitrary topologies. In *AAMAS'09*, 2009. 6.3

[8] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997. 4.4

[9] John R. Birge and François Louveaux. *Introduction to Stochastic Programming*. Springer Series in Operations Research and Financial Engineering. Springer, July 1997. 7.2.2

[10] S. Bohacek, J.P. Hespanha, and K. Obraczka. Saddle policies for secure routing in communication networks. In *Decision and Control, 2002*, 2002. 6.3, 6.3

[11] Amit Chakrabarti, Chandra Chekuri, Anupam Gupta, and Amit Kumar. Approximation algorithms for the unsplittable flow problem. *Algorithmica*, 47(1):53–78, 2007. 6.1

[12] Mainak Chatterjee, Sajal K. Das, and Damla Turgut. WCA: A weighted clustering algorithm for mobile ad hoc networks. *Journal of Cluster Computing (Special Issue on Mobile Ad hoc Networks*, 5:193–204, 2001. 6.2

[13] Vincent Conitzer and Tuomas Sandholm. Computing the optimal strategy to commit to. In *Proceedings of the 7th ACM conference on Electronic commerce*, EC '06, pages 82–90, New York, NY, USA, 2006. ACM. 6.3

[14] Leon Cooper. The transportation-location problem. *Operations Research*, 20:94–108, 1972. 6.2

[15] George B. Dantzig. Linear programming under uncertainty. *Management Science*, 1(3-4):197–206, 1955. 7.2.2

[16] K. Decker and V. R. Lesser. Quantitative Modeling of Complex Environments. *International Journal of Intelligent Systems in Accounting, Finance and Management. Special Issue on Mathematical and Computational Models and Characteristics of Agent Behaviour.*, 2:215–234, January 1993. 1.1

[17] Pavlos S. Efraimidis and Paul G. Spirakis. Weighted random sampling with a reservoir. *Information Processing Letters*, 97(5):181 – 185, 2006. 4.5

[18] Kutluhan Erol, James Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94*, pages 1123–1128. AAAI Press, 1994. 1.1

[19] V. Erramilli, I. Malta, and A. Bestavros. On the interaction between data aggregation and topology control in wireless sensor networks. pages 557 – 565, oct. 2004. 6.1

[20] Shimon Even. *Graph Algorithms*. Computer Science Press, Rockville, Maryland, 1979. 3.1

[21] M. R. Garey and D. S. Johnson. "Strong" NP-completeness results: Motivation, examples, and implications. *Journal of the ACM*, 25(3):499–508, July 1978. 2.1, 2.3

[22] Brian P. Gerkey and Maja J. Mataric. Sold!: auction methods for multirobot coordination. *IEEE Transactions on Robotics*, 18(5):758–768, 2002. 1.1

[23] Mario Gerla and Jack Tzu chieh Tsai. Multicluster, mobile, multimedia radio network. *Journal of Wireless Networks*, 1:255–265, 1995. 6.2

[24] Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269 – 357, 1996. 1.1

[25] Anupam Gupta, Jon Kleinberg, Amit Kumar, Rajeev Rastogi, and Bulent Yener. Provisioning a virtual private network: A network design problem for multicommodity flow. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, 2001. 6.2

[26] Venkatesan Guruswami, Sanjeev Khanna, Rajmohan Rajaraman, Bruce Shepherd, and Mihalis Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. *Journal of Computer and System Sciences*, 67:473–496, November 2003. 2.3, 6.1

[27] E. Halvorson, V. Conitzer, and R. Parr. Multi-step Multi-sensor Hider-Seeker Games. In *IJCAI'09*, 2009. 6.3

[28] Wendi Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS '00*, 2000. 2.2, 6.1

[29] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw.*, 11(1):2–16, 2003. 2.2, 6.1

[30] Eitan Israeli and R. Kevin Wood. Shortest-path network interdiction. *Networks*, 40:97–111, 2002. 6.3

[31] Kamal Jain, Jitendra Padhye, Venkata N. Padmanabhan, and Lili Qiu. Impact of interference on multi-hop wireless network performance. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, MobiCom '03, pages 66–80, New York, NY, USA, 2003. ACM. 2.1.1

[32] Manish Jain, Dmytro Korzhyk, Ondrej Vanek, Vincent Conitzer, Michal Pechoucek, and Milind Tambe. A double oracle algorithm for zero-sum security games on graphs. In *International Conference on Autonomous Agents and Multiagent Systems*, 2011. 6.3

[33] Raja Jothi and Balaji Raghavachari. Approximation algorithms for the capacitated minimum spanning tree problem and its variants in network design. *ACM Trans. Algorithms*, 1(2):265–282, 2005. 6.1

[34] Christopher Kiekintveld, Manish Jain, Jason Tsai, James Pita, Fernando Ordez, and Milind Tambe. Computing optimal randomized resource allocations for massive security games. In *AAMAS'09*, 2009. 5.7.1, 6.3

[35] Hiroaki Kitano, Satoshi Tadokoro, Itsuki Noda, Hitoshi Matsubara, Tomoichi Takahashi, Atsuhi Shinjou, and Susumu Shimada. Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In *IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS*, pages 739–746. IEEE Computer Society, 1999. 1.1

[36] J. M. Kleinberg. Single-source unsplittable flow. In *FOCS '96: Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, page 68, Washington, DC, USA, 1996. IEEE Computer Society. 6.1

[37] V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. NagendraPrasad, A. Raja, R. Vincent, P. Xuan, and X.Q Zhang. Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143, July 2004. 1.1

[38] Ning Li and Jennifer C. Hou. Improving connectivity of wireless ad hoc networks. In *MobiQuitous'05*, 2005. 1.3

[39] S. Lindsey and C.S. Raghavendra. PEGASIS: Power-efficient gathering in sensor information systems. volume 3, pages 3–1125 – 3–1130 vol.3, 2002. 6.1

[40] Robert E. Love, James G. Morris, and George O. Wesolowsky. *Facilities Location*. North Holland, New York, 1988. 6.2

[41] Roger Mailler and Victor Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS'04*, 2004. 1.1

[42] M Mavronicolas, V. Papadopoulou, A. Philippou, and P. Spirakis. A network game with attackers and a defender. *Operation Research*, 43(2):243–251, 1995. 6.3

[43] H Brendan McMahan, Geoffrey J Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *Proceedings of the Twentieth International Conference on Machine Learning*, 2003. 6.3

[44] H. Moulin and J. P. Vial. Strategically zero-sum games: The class of games whose completely mixed equilibria cannot be improved upon. *International Journal of Game Theory*, 7:201–221, 1978. 10.1007/BF01769190. 6.3

[45] Robert M. Nauss. An improved algorithm for the capacitated facility location problem. *The Journal of the Operational Research Society*, 29:1195–1201, December 1978. 6.2

[46] Steven Okamoto, Noam Hazon, and Katia Sycara. Solving non-zero sum multiagent network flow security games with attack costs. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '12, pages 879–888, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems. 5

[47] Steven Okamoto, Praveen Paruchuri, Yonghong Wang, Katia Sycara, Janusz Marecki, and Mudhakar Srivatsa. Multiagent communication security in adversarial settings. In *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Volume 02*, WI-IAT '11, pages 296–303, Washington, DC, USA, 2011. IEEE Computer Society. 4

[48] Steven Okamoto and Katia Sycara. Augmenting ad hoc networks for data aggregation and dissemination. In *Proceedings of the 28th IEEE conference on Military communications*, MILCOM'09, pages 2346–2352, Piscataway, NJ, USA, 2009. IEEE Press. 3

[49] Jianping Pan, Y. Thomas Hou, Lin Cai, Yi Shi, and Sherman X. Shen. Topology control for wireless sensor networks. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 286–299, New York, NY, USA, 2003. ACM. 2.2, 6.1

[50] P. Paruchuri, J.P. Pearce, J. Marecki, M. Tambe, F. Ordoñez, and S. Kraus. Playing games with security: An efficient exact algorithm for Bayesian Stackelberg games. In *AAMAS'08*, 2008. 6.3

[51] Adrian Petcu, Boi Faltings, and Roger Mailler. PC-DPOP: a new partial centralization algorithm for distributed optimization. In *Proceedings of the 20th international*

joint conference on Artifical intelligence, IJCAI'07, pages 167–172, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. 1.1

[52] James Pita, Manish Jain, Janusz Marecki, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Deployed ARMOR protection: the application of a game theoretic model for security at the Los Angeles International Airport. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, AAMAS '08, pages 125–132, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems. 6.3

[53] David V. Pynadath, Milind Tambe, and Nicolas Chauvat. Toward team-oriented programming. In *Intelligent Agents VI: Agent Theories, Architectures, and Languages*, pages 233–247. Springer-Verlag, 1999. 1.1

[54] R. Ravi and A. Sinha. Multicommodity facility location. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '04, pages 342–349, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics. 6.2

[55] Alissa J. Rubin. Attack at Kabul hotel deflates security hopes in Afghanistan. New York Times, June 29 2011. 5.3

[56] F. Sibel Salman, R. Ravi, and John N. Hooker. Solving the capacitated local access network design problem. *INFORMS Journal on Computing*, 2008. 6.2

[57] Paul Scerri, Elizabeth Liao, Y. Xu, Michael Lewis, G. Lai, and Katia Sycara. Coordinating very large groups of wide area search munitions. *Theory and Algorithms for Cooperative Systems*, 2005. 1.1

[58] Paul Scerri, D. Pynadath, N. Schurr, A. Farinelli, S. Gandhe, and M. Tambe. Team oriented programming and proxy agents: The next generation. In *Proceedings of 1st international workshop on Programming Multiagent Systems*, 2004. 1.1

[59] Bernhard Von Stengel and Shmuel Zamir. Leadership with commitment to mixed strategies. Technical report, London School of Economics, 2004. 5.7.1, 6.3

[60] Katia Sycara, Massimo Paolucci, Martin Van Velsen, and Joseph Andrew Giampapa. The RETSINA MAS infrastructure. *Autonomous Agents and Multi-Agent Systems*, 7(1/2):29–48, July 2003. 1.1

[61] Jian Tang, Bin Hao, and Arunabha Sen. Relay node placement in large scale wireless sensor networks. *Computer Communications*, 29(4):490 – 501, 2006. Current areas of interest in wireless sensor networks designs. 1.3

[62] J. Tsai, Z. Yin, J. Kwak, D. Kempe, C. Kiekintveld, and M. Tambe. Urban security: Game-theoretic resource allocation in networked physical domains. In *AAAI'10*, 2010. 4.5, 4.6.1, 6.3, 6.3

[63] Jason Tsai, Thanh H. Nguyen, and Milind Tambe. Security games for controlling contagion. In *Conference on Artificial Intelligence (AAAI)*, 2012. 6.3

[64] Jason Tsai, Shyamsunder Rathi, Christopher Kiekintveld, Fernando Ordez, and Milind Tambe. IRIS - a tool for strategic security allocation in transportation networks. In *The Eighth International Conference on Autonomous Agents and Multiagent Systems - Industry Track*, 2009. 6.3

[65] Alex Varshavsky and Eyal de Lara. Alleviating self-interference in manets. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, LCN '04, pages 642–649, Washington, DC, USA, 2004. IEEE Computer Society. 2.1.1

[66] A. Washburn and K Wood. Two-person zero-sum games for network interdiction. *Operation Research*, 43(2):243–251, 1995. 4.3, 6.3

[67] George O. Wesolowsky. Dynamic facility location. *Management Science*, 19:1241–1248. 6.2

[68] Kaixin Xu, Xiaoyan Hong, and M. Gerla. An ad hoc network with mobile backbones. In *Communications, 2002. ICC 2002. IEEE International Conference on*, volume 5, pages 3138–3143 vol.5, 2002. 1.3

[69] Zhengyu Yin, Dmytro Korzhyk, Christopher Kiekintveld, Vincent Conitzer, and Milind Tambe. Stackelberg vs. Nash in security games: Interchangeability, equivalence, and uniqueness. In *AAMAS'10*, 2010. 6.3, 6.3