8-2018

Mathematical Models and Algorithms for Network Flow Problems Arising in Wireless Sensor Network Applications

Robert M. Curry
Clemson University, rmcurry@clemson.edu

Mathematical Models and Algorithms for Network Flow Problems Arising in Wireless Sensor Network Applications

---

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Industrial Engineering

---

by
Robert M. Curry
August 2018

Accepted by:
Dr. J. Cole Smith, Committee Chair
Dr. Warren Adams
Dr. Sandra D. Eksioglu
Dr. Amin Khademi

Abstract

We examine multiple variations on two classical network flow problems, the maximum flow and minimum-cost flow problems. These two problems are well-studied within the optimization community, and many models and algorithms have been presented for their solution. Due to the unique characteristics of the problems we consider, existing approaches cannot be directly applied. The problem variations we examine commonly arise in wireless sensor network (WSN) applications. A WSN consists of a set of sensors and collection sinks that gather and analyze environmental conditions. In addition to providing a taxonomy of relevant literature, we present mathematical programming models and algorithms for solving such problems.

First, we consider a variation of the maximum flow problem having node-capacity restrictions. As an alternative to solving a single linear programming (LP) model, we present two alternative solution techniques. The first iteratively solves two smaller auxiliary LP models, and the second is a heuristic approach that avoids solving any LP. We also examine a variation of the maximum flow problem having semicontinuous restrictions that requires the flow, if positive, on any path to be greater than or equal to a minimum threshold. To avoid solving a mixed-integer programming (MIP) model, we present a branch-and-price algorithm that significantly improves the computational time required to solve the problem.

Finally, we study two dynamic network flow problems that arise in wireless sensor networks under non-simultaneous flow assumptions. We first consider a dynamic maximum flow problem that requires an arc to transmit a minimum amount of flow each time it begins transmission. We present an MIP for solving this problem along with a heuristic algorithm for its solution. Additionally, we study a dynamic minimum-cost flow problem, in which an additional cost is incurred each time an arc begins transmission. In addition to an MIP, we present an exact algorithm that iteratively solves a relaxed version of the MIP until an optimal solution is found.

Acknowledgments

All thanks and praise to God for his grace, mercy, and provision. I hope my work will always reflect his goodness and creativity. Thank you to my best friend and wonderful wife, Lauren. Without her support, I would never have completed my Ph.D. She has sacrificially helped and cared for me during these last four years. I cannot express enough thanks to my friend, mentor, and advisor, Dr. Cole Smith, for his guidance and wisdom. Countless hours discussing research and writing have made me a better researcher and human being. I am also thankful for the countless miles run together discussing everything under the sun. Thank you to the rest of my committee for helping me to hone my research skills and ideas through my coursework, comprehensive exam, and dissertation defense. To my parents, thank you for your continued love and encouragement. This was made possible because of your hard work and sacrifice. Finally, thank you to all the students and faculty that have supported and accompanied me during my academic journey.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The mathematical optimization community has long examined network flow problems that take place on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ with a node set $\mathcal{V}$ and an arc set $\mathcal{A}$. Set $\mathcal{V}$ contains a source node s and a sink node t. We study variations on two of the most common problems in this area, the maximum flow problem and the minimum-cost flow problem. Maximum flow problems involve finding a feasible flow through a single-source, single-sink flow network that is maximum, and minimum-cost flow problems involve finding a feasible flow that minimizes the cost of transmitting a predetermined amount of flow from s to t. Classic versions of these problems are well-studied within the optimization communities, and many models and algorithms have been presented for their solution [Ahuja et al., 1993]. Problems of this sort commonly arise in *wireless sensor network* (WSN) optimization. These networks consist of a set of sensors and collection sinks that gather and analyze environmental conditions. The emergence of WSN optimization uncovered the need for modeling and solving adaptations of these classic network flow problems existing in WSNs.

1.1 Background and Contribution

This dissertation presents models and algorithms for solving various classes of network flow problems having applications in WSN optimization. Wireless sensor network research has recently gained substantial attention in the engineering, computer science, and mathematics literature because of the challenges involved in deploying effective networks in energy- and bandwidth-constrained settings. WSNs incorporate battery-powered sensors that expend energy to collect, analyze, and

transmit information. Where and how these sensors are placed may depend on environmental characteristics, the accessibility of the environment, forces that relocate the sensors, and factors that potentially impede the operation of those sensors. Prior to this work, the optimization literature lacked a full understanding of the problems in this application area. Thus, we first present a comprehensive survey of applications, problems, and solution techniques for solving maximum flow-related problems in WSN settings.

Sensors in WSNs commonly possess budget restrictions in the form of finite-capacity batteries used to transmit data to other sensors or a grounded network collection location. Replacing these batteries can be difficult or hazardous because WSNs are often deployed in unreachable or dangerous environments (e.g., military applications [Abbasi and Younis, 2007] and environmental monitoring [Xu et al., 2014]). Thus, we first examine the problem of maximizing the total amount of data gathered in the network without exceeding battery power or bandwidth restrictions. This problem can be formulated as the maximum flow problem having node-capacity restrictions [Kalpakis et al., 2003, Curry and Smith, 2016]. Environmental restrictions in large-scale energy systems can also be modeled as node-capacity constraints. These restrictions include a maximum emission level for generation stations, where the amount of emissions may depend on the type of fuel used, pollution limiting devices utilized, and the amount of energy produced [Quelhas et al., 2007]. Congestion restrictions in large-scale virtual channel infrastructure systems can also be modeled as a variation of node-capacity constraints [Peltola et al., 1997].

We model this problem as a variation of the maximum flow problem (MFP) having a set of node-capacity restrictions (NCMFP). Not surprisingly, the maximum flow-minimum cut theorem [Ahuja et al., 1993] for the MFP does not hold for the NCMFP. Thus, traditional augmenting-flow algorithms (e.g., Ford-Fulkerson [Ford and Fulkerson, 1956] and push-relabel [Ahuja et al., 1993]) generally will not optimize the NCMFP. One of the key observations is that an analogous criterion — that every such path *either* visits a saturated arc *or* a node whose capacity is exhausted — is not sufficient to prove optimality. Generally speaking, there may exist a flow circulation that increases the available capacity on a set of exhausted nodes, thus permitting additional flow to be transmitted from s to t.

The second problem we study is a variation of the MFP having node and arc capacities, along with semicontinuous flow restrictions. A semicontinuous variable must either take a value of 0, or belong in the interval $[\ell, u]$ for some $0 < \ell \leq u$. We assume that the upper bounds are implied by

the arc-flow capacities, and thus we only focus on how to handle the lower bound restrictions on positive flows. In the context of flow problems, semicontinuous restrictions are useful when positive flows below some lower bound are undesirable.

However, a precise characterization of semicontinuous flows depends on what one considers to be the flow variables. We examine several such cases in this work. Throughout, flows that satisfy semicontinuous restrictions with respect to a given set of variables are said to be *stable*. The simplest case enforces stability restrictions on arc flows, which can be achieved by defining a binary variable $y_{ij},\ \forall (i,j) \in \mathcal{A}$, and restricting $uy_{ij} \geq x_{ij} \geq \ell y_{ij}$ [Beale, 1979, 1980, 1985]. The work of de Farias et al. [2001] proposes an alternative branching strategy to address semicontinuity. By contrast, we examine problems having stability restrictions on network paths, in which a *path* refers to a common amount of flow from any source node to any demand node using a sequence of arcs that visits each node at most once. Assuming all flows are simultaneously transmitted, we require that flow f_p on each path p, if positive, must be greater than or equal to ℓ.

Finally, we consider network flow problems in dynamic network flow settings, in which flows are transmitted according to a non-simultaneous schedule. Specifically, we consider a pair of dynamic network flow problems that consider the existence of arc setup constraints or costs that may exist whenever an arc begins transmitting flow. An arc may be required to undergo setup each time it begins transmitting flow. Alternatively, some applications have explicit setup costs, in which a fixed cost is incurred each time an arc begins transmission. In WSN optimization, operators may need to pay a fixed cost to begin transmitting information from one sensor to another [Yick et al., 2008]. These fixed costs often correspond to the financial and/or computational effort required to establish a secure communication link between a pair of sensors [Perrig et al., 2004, Shi and Perrig, 2004]. Whenever a sensor discontinues transmission, this communication link either ceases to exist or is assumed not to be secure. Thus, each time a sensor begins transmission, a new secure communication link must be set up to avoid interference.

1.2 Literature Review and Application Areas

The arc-capacitated MFP has many areas of application (e.g., transportation problems, airline scheduling, project selection [Ahuja et al., 1993]). Ford and Fulkerson [1956] first formulate the MFP as a linear programming (LP) model and show its equivalence to the minimum-cut problem.

(See [Schrijver, 2002] for a comprehensive history of the maximum flow problem.) They present a pseudo-polynomial algorithm that augments flow on a series of paths from s to t having positive residual capacity. Edmonds and Karp [1972] later propose a polynomial-time path-augmenting algorithm that implements a shortest-path procedure to find augmenting paths. Ahuja and Orlin [1989] introduce a capacity-scaling algorithm for solving the MFP in polynomial time that prioritizes flow augmentations on paths containing only those arcs having residual capacity levels greater than or equal to some minimum threshold.

Whereas the previous approaches employ path augmentation, others extend alternative techniques for solving the MFP. Goldfarb and Hao [1990] propose a strongly-polynomial primal simplex algorithm for the MFP and give implementations that run in $O(|\mathcal{V}|^2|\mathcal{A}|)$ time. Alternatively, Goldfarb and Chen [1997] and Armstrong et al. [1998] take a dual simplex-based approach to the MFP. Ahuja and Orlin [1997] show the equivalence of these primal and dual simplex approaches. More recently, Hochbaum [2008] shows how the MFP can be transformed into a variation of the minimum-cost flow problem, and presents a pseudo-flow algorithm for its solution. For a comprehensive analysis of algorithmic advances for the MFP, see [Ahuja et al., 1991] and [Cook et al., 1998].

Wollmer [1968] presents a polynomial-time flow-augmenting algorithm for the NCMFP when the capacity consumed by sending a unit of flow at a node is assumed to be binary. The author presents an LP formulation as well as a polynomial-time algorithm for its solution. In WSN settings, Chang and Tassiulas [1999] model the problem of maximizing the amount of gathered data as the NCMFP. The authors present an LP along with two heuristic algorithms for its solution. Since the NCMFP is a variation of the MFP having a set of complicating constraints, Madan and Lall [2004] employ a Lagrangian relaxation technique having Lagrangian multipliers associated with the node-capacity constraints. Ordóñez and Krishnamachari [2004] formulate a nonlinear programming model to maximize the total amount of data gathered by a wireless sensor network (WSN) when sensors have a maximum transmission capacity. Bodlaender et al. [2008] consider a problem similar to ours, in that they include node-capacity constraints for the maximum flow problem (but without semicontinuity restrictions). The authors maximize the total data gathered among all sensors in the network and restrict flows to be integer-valued, as would be the case when sensors transmit data packets in their entirety. Chapter 2 provides a detailed analysis of such problems in WSN optimization.

Semicontinuous restrictions arise in several different application areas. Bienstock [1996] and

Perold [1984] use semicontinuous variables to model minimum trading sizes in portfolio optimization. In inventory management models, Timpe and Kallrath [2000] require order shipments to be between some minimum and maximum quantities. Kallrath [2000, 2002] considers the presence of semicontinuous variables in petrochemical processes. Angulo et al. [2014] examine a relaxation of general semicontinuous network flow problems. The authors present a complete description of the convex hull with linear inequalities and extended formulations for two particular cases of this relaxation.

Static-stable restrictions arise in WSN settings, in which network operators attempt to gather as much information about an environment as possible. This problem can be modeled as an instance of the maximum flow problem since the objective may seek to collect and transmit the maximum possible amount of data. Information gathered by any sensor can be transmitted to any sink node, which then relays its data to a common source (e.g., a single satellite) for analysis. Tao et al. [2004] consider a related problem where the flow on each path from s to t is determined at a central location within the environment. The authors include restrictions that disallow paths having flow below a minimum threshold, because switching between s–t paths for small improvements in the objective function value is impractical.

Additionally, stability restrictions arise in scheduling applications. Aggarwal and Orlin [2002] model a machine scheduling problem as an instance of the maximum flow problem (without semicontinuity restrictions). A slight generalization of their problem is as follows. Let $\mathcal{J}$ be a set of jobs, $\mathcal{M}$ be a set of machines, and $\mathcal{W}$ be a set of workers. Let $\mathcal{M}(j)$ be the set of machines on which job j can be processed, and let $\mathcal{W}(m)$ be the set of workers that can process any job on machine m. Each job $j \in \mathcal{J}$ has a required processing time, p_j, and each worker $w \in \mathcal{W}$ can perform no more than h_w total hours of work. The processing speed for each job does not depend on the worker or the machine. This problem can be on a graph $\mathcal{G}' = (\mathcal{V}', \mathcal{A}')$, using the concepts from [Aggarwal and Orlin, 2002]. Set $\mathcal{V}'$ contains source s, sink t, and nodes corresponding to $\mathcal{J}$, $\mathcal{M}$, and $\mathcal{W}$. Set $\mathcal{A}'$ contains arcs $(s, j),\ \forall j \in \mathcal{J}$; arcs $(j, m),\ \forall j \in J$ and $m \in \mathcal{M}(j)$; arcs $(m, w),\ \forall m \in \mathcal{M}$ and $w \in \mathcal{W}(m)$; and arcs $(w, t),\ \forall w \in \mathcal{W}$. The flow on any path (s, j, m, w) represents the time worker w spends processing job j on machine m. The required processing times p_j for each $j \in \mathcal{J}$ provide capacities on each arc $(s, j) \in \mathcal{A}'$, and the worker processing limits h_w, $w \in \mathcal{W}$, serve as the capacities for each arc $(w, t) \in \mathcal{A}'$. All other arcs are uncapacitated. Solving the MFP thus determines how much of the total required processing time over all jobs can be scheduled on the machines using the available resources.

For some cases of this problem, a worker may require added effort when switching to a new job or machine. This additional effort is often undesirable for many reasons, including safety and quality considerations that arise when workers switch tasks (see, e.g., [Monsell, 2003]), or extra costs that might be incurred in switching tasks. Practical guidelines would require each worker to spend a minimum amount of time processing job j on machine m before switching to a new job or machine. Our static-stable restrictions in Chapter 4 model this policy, since each path corresponds to a unique combination of job j being processed on machine m by worker w. A slight modification occurs when jobs that can be performed on a common machine are so similar that the cognitive and physical effort for a worker to switch from one job to another is negligible, although the effort to switch work on one machine to another remains significant. A minimum processing time policy in this case can be modeled by a set of dynamic-stable restrictions placed on each arc $(m, w) \in \mathcal{A}'$ to ensure that a worker spends at least ℓ units of time processing jobs on machine m.

Finally, problems in dynamic network flow problems commonly arise in time-dependent network flow settings. For example, the dynamic flow problem having setup costs also models network flow problems in interdependent infrastructure systems. Cavdaroglu et al. [2013] consider the problem of restoring public services after a disaster disrupts civil infrastructure systems. In these problems, the managers pay a fixed financial or labor cost to deploy a set of temporary arcs that provide an interim infrastructure for deploying services (e.g. transportation, telecommunication, and power). Since these infrastructures are commonly interdependent, restoration and planning decisions must be made according to non-simultaneous schedule. Therefore, the dynamic flow problem having setup costs could be employed to model a variation of this problem, in which temporary arcs expire after infrastructures stop using them.

1.3 Dissertation Organization

Chapter 2 first includes a comprehensive survey of maximum flow-related problems in WSN settings. Furthermore, WSN optimization problems also consider alternative metrics (e.g., coverage area, connectivity, and transmission delay) in addition to maximum flow. Our survey also discusses these various optimization challenges and describes some of the proposed algorithms currently existing in the literature. The open topics within the literature found while working on this survey led to the problems considered and techniques employed in the latter chapters of this work.

In Chapter 3, we present two augmenting-flow algorithms that avoid solving a large LP for the NCMFP. Both algorithms modify the augmenting-path algorithm to obtain a feasible NCMFP solution. The first algorithm implements two smaller auxiliary LPs to solve the NCMFP. These two LPs augment either prove the optimality of the current NCMFP solution, terminate with a feasible NCMFP solution, or augment flow first on a circulation to increase some node capacities and then on an s–t path to maximize additional flow in the network. This first algorithm is an almost-exact solution technique and tends to encounter small, easily solvable LPs. In a majority of cases, this technique obtains an optimal NCMFP solution. In other cases, this technique produces quality solutions to the NCMFP that are close to optimal. However, some situations might call for the complete avoidance of LP subroutines. Accordingly, our second approach is heuristic only, and modifies the circulation-generation part of the first approach to obviate the necessity of solving an LP.

Chapter 4 details models and algorithms for solving the MFP having semicontinuous flow restrictions on paths in simultaneous flow settings. We first establish the relationship between the feasible regions of arc-based and path-based stability restrictions. As opposed to solving a mixed-integer programming (MIP) model, we propose a branch-and-price algorithm for this problem, including a specialized branching strategy that leverages the existence of cut-sets in a non-feasible solution. Additionally, we highlight a special type of cut-set that allows for stronger branching inequalities. We finally detail the efficacy of our algorithm on a randomly-generated set of test networks.

We finally study a pair of dynamic network flow problems in Chapter 5. The first problem considers the presence of setup requirements, and the second considers the presence of setup costs. The former can be modeled by the MFP having dynamic stability restrictions (MFP-D). The motivation behind dynamic stability is that when an arc is used to send flow, it sends at least ℓ units of flow in an uninterrupted interval of time. The latter problem can be modeled by the minimum-cost flow problem having *arc-activation* costs (MCF-A), in which an arc (i, j) is said to be *activated* on path p when (i, j) has positive flow on the p^{th} scheduled path, but not on the $(p-1)^{st}$ scheduled path. We first introduce the notion of dynamic stability as well as an MIP model for the MFP-D. We present a heuristic algorithm that obtains lower and upper bounds for the MFP-D. We then provide motivation for the MCF-A and present an MIP model for its solution. As an alternative to this MIP, we employ a relaxation-based algorithm for obtaining upper and lower bounds that increases the

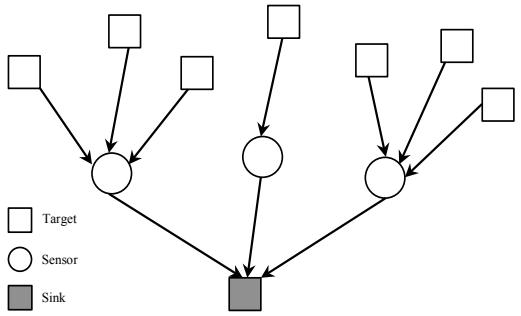
number of paths in an MCF-A schedule, as needed.

Chapter 2

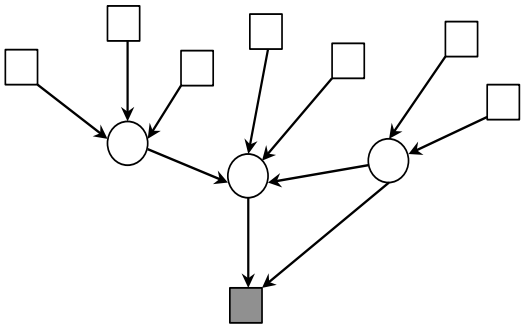
Survey of Maximum Lifetime Maximization Problems in WSN Applications

2.1 Introduction

We consider a general WSN having a set of target nodes, sensor nodes, and a single sink node (Figure 2.1). Targets are physical locations within the environment where the conditions of interest originate. Information found at these targets must be monitored and retrieved by a sensor. Upon retrieval, sensors are tasked with sending this data to the sink for collection and processing. The WSN sink then aggregates and processes environmental information retrieved by sensors. Sensors can transmit data directly to the sink, as depicted in Figure 2.1a. Alternatively, sensors can transmit data to the sink via multi-hop communication, where sensors relay data through neighboring sensors en route to the sink, as shown in Figure 2.1b [Pottie and Kaiser, 2000, Raghavendra et al., 2004]. Note that in WSNs employing multi-hop communication, a significant amount of data flows through a relatively small set of sensors located near the sink, which may consume their battery prematurely. This behavior is called the *energy-hole problem* [Li and Mohapatra, 2005, 2007], and is typical of WSNs employing multi-hop communication. Whereas the energy required for a sensor to receive data may only be a function of the amount received, energy required to transmit data to an object (such



(a) WSN direct communication



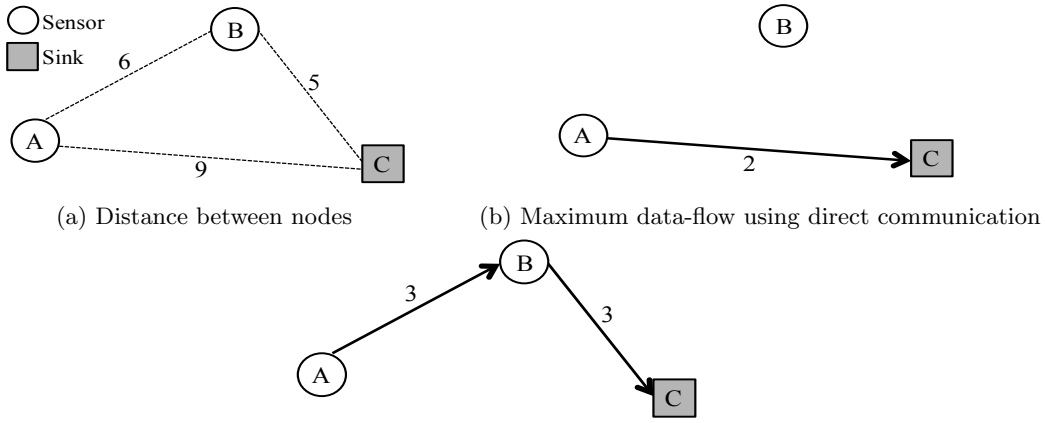
(b) WSN multi-hop communication

Figure 2.1: Wireless sensor network routing

as another sensor or sink) predominantly depends on both the amount of data transmitted and the distance between the sensor and that object. Each sensor possesses finite battery power to handle these expenditures. Because WSNs can be used to, e.g., observe adversarial movement, monitor weapons testing, and detect seismic activity [Dutta et al., 2010, Min et al., 2012, Werner-Allen et al., 2006], sensors often dwell in hostile or inaccessible areas where battery replacement is impractical or prohibitively costly. When the battery is fully consumed, a sensor is no longer able to monitor targets, thus rendering the network inoperative. As a result, many studies address maximizing WSN lifetime, defined as the first time at which a sensor's battery is exhausted [Basagni et al., 2009, Behdani et al., 2012, Chang and Tassiulas, 1999, 2004, Keskin et al., 2011, Gatzianas and Georgiadis, 2008]. Most WSN studies omit the targets in our problem setting and assume that the sensors are assigned *a priori* to targets. We will adopt that assumption in this chapter unless otherwise specified.

Multi-hop communication is often employed to prolong network lifetime by balancing energy consumption across all sensors. To illustrate this balancing mechanism and its benefits, the small example in Figure 2.2 depicts a network having two sensors, A and B, and a single sink, C. The goal is to transmit data from A to C as long as possible, with B serving solely as a relay node where data

is not retrieved. In this example, each unit of data transmitted a distance of d consumes d units of energy at the transmitting sensor and one unit of energy at the receiving sensor. The distances from A to B, A to C, and B to C are 6, 9, and 5 units, respectively (Figure 2.2a). Suppose that batteries at sensors A and B possess 18 total units of energy each. Using direct communication, sensor A would only be able to send two units of data directly to C, while B would not consume any battery power (Figure 2.2b). On the other hand, suppose data is sent from A to B, and then from B to C, as in Figure 2.2c. Each transmitted unit consumes six units of energy at A, while B consumes one unit to receive the data and five more to relay it to C. Thus, while only two units can be transmitted directly from A to C, three units of data can be transmitted on the A-B-C path. The latter route requires more total energy, but more importantly balances energy consumption to extend the network lifetime.



(a) Distance between nodes

(b) Maximum data-flow using direct communication

(c) Maximum data-flow using multi-hop communication

Figure 2.2: WSN multi-hop communication example

Due to their low cost and practical importance, sensor applications arise in many diverse areas. Three examples of these areas follow.

- The advent of *body area networks* (BANs) promises to allow physicians to monitor individual health on a real-time basis. A BAN is a type of WSN that consists of a networked group of sensors located in a non-invasive or invasive manner on the body, tracking items such as sleep habits, blood glucose, blood pressure, and EEG patterns [Chen et al., 2011, Ko et al., 2010]. Of interest in this chapter are sensors that can be implanted in the body, for whom battery conservation is more critical. Movassaghi et al. [2014] discuss opportunities for implanted BANs to monitor diabetes, cardiovascular disease, and cancer detection and spread.

- The ability to monitor or provide surveillance in military applications clearly relies on WSN deployment and operations. Lee et al. [2009] describe several defense scenarios. Some involve the deployment of sensors on equipment or personnel in the field in order to coordinate operations and avoid casualties due to friendly fire. Others place sensors behind enemy lines, forcing them to self-organize and conserve battery utilization. These sensors can be ground-, sea-, or air-based in practice.

- Environmental monitoring is a third primary application area for WSN deployment. Lundquist et al. [2003] describe an application in which sensors are used to monitor meteorological and hydrologic processes in the Sierra Nevada mountain range. This landscape is difficult to traverse, and so it becomes advantageous to use sensors to monitor conditions in these areas. A similar project is discussed in the Swiss Alps, in which the researchers seek to detect conditions under which avalanches will occur [Barrenetxea et al., 2008], in addition to the same types of data sought in [Lundquist et al., 2003]. The inaccessibility of these environment and dangers associated with avalanches justify the need for WSN use and the conservation of sensor batteries.

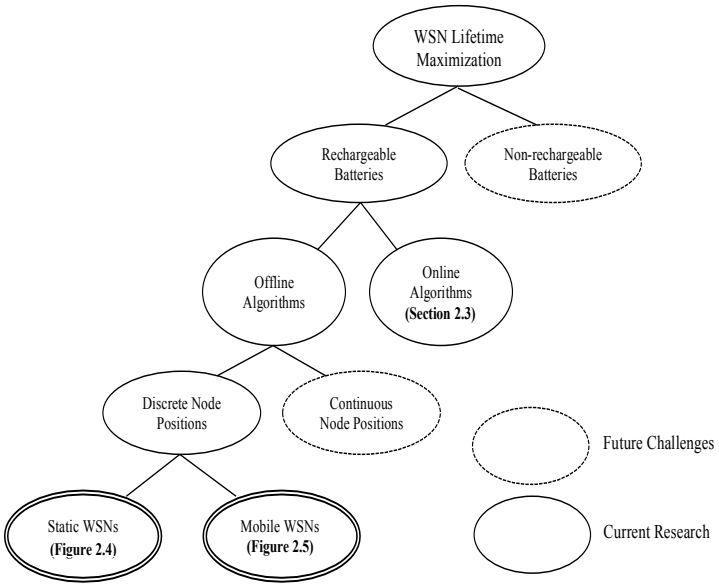


Figure 2.3: WSN taxonomy

Some WSNs group sensors into clusters to decentralize and simplify network coordination. Rather than randomly deploying sensors, some WSNs assume a specific topology, such as a uniform

two-dimensional grid to simplify routing decisions. Other application areas allow sensors to fuse data gathered from multiple targets before transmission to the sink. By doing so, the size of data transmission packets are reduced to lessen energy consumption. Additionally, sinks may be able to move about the sensing area to balance energy consumption. In such cases, sensors may locally buffer data to delay transmission until the moving sink arrives at a more favorable location. Figures 2.3, 2.4, and 2.5 display a taxonomy of WSN lifetime maximization research to describe various attributes of and interrelationships between the problems we address.

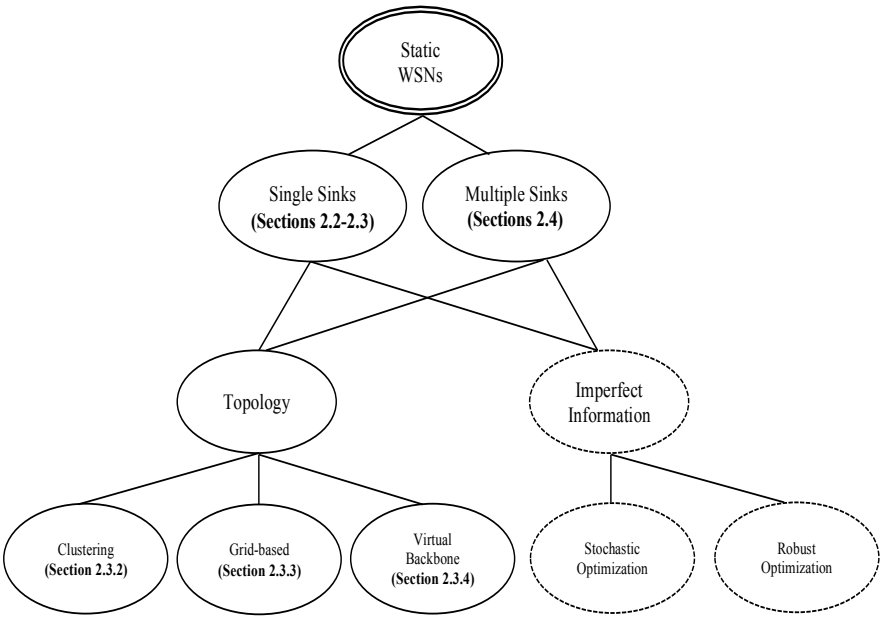


Figure 2.4: Static WSN taxonomy

Because of the recent attention to WSNs, other papers have reviewed a portion of the research literature in this field. Yick et al. [2008] and Rawat et al. [2014] provide a broad overview of fundamental WSN research, in which they highlight various application areas, in addition to standard operating systems, communication protocols, and network services. Other review articles focus on algorithms and mathematical modeling approaches for WSN optimization. Anastasi et al. [2009] and Rault et al. [2014] review papers that examine WSN design for various application areas. Another set of articles review WSN routing protocols. Akkaya and Younis [2005] focus on hierarchical and location-based approaches for routing, while Pantazis et al. [2013] consider communication models, topology, and reliability when making routing decisions. Additionally, Yu et al. [2014], Tunca et al. [2014], and Sara and Sridharan [2014] survey routing protocols and deployment techniques

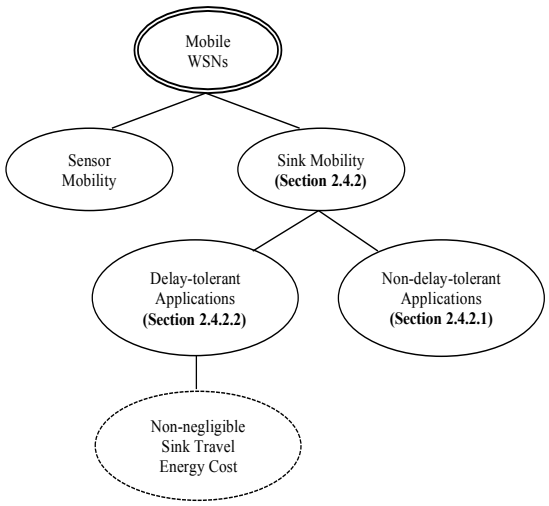


Figure 2.5: Mobile WSN taxonomy

for mobile-sink WSNs. Whereas these works primarily focus on algorithms for energy consumption minimization, this chapter concentrates on algorithms and mathematical models for maximizing WSN lifetime.

Rather than minimizing energy consumption, Younis et al. [2014] and Li et al. [2013] explore topology design and control techniques for maximizing coverage and connectivity. Other authors review clustering approaches for improving network scalability ([Abbasi and Younis, 2007], [Afsar and Tayarani-N., 2014], [Schaffer et al., 2012]). These papers review algorithms that focus on various objectives independent of WSN lifetime. Alternatively, we also give a broader overview of topology-based approaches that focus on maximizing WSN lifetime.

The rest of the chapter is organized as follows. In Section 2.2 we present notation and solution techniques for the fundamental WSN lifetime maximization problem. We discuss online routing, clustering techniques, and lifetime maximization on special structures in Section 2.3. Section 2.4 then highlights extensions to the fundamental problem that result from having multiple sinks, which may also be mobile instead of stationary. We review problems and algorithms that consider alternative optimization metrics to lifetime maximization in Section 2.5. Finally, we examine some of the ongoing and future challenges facing WSN optimization in Section 2.6. Summary tables appear in the beginning of each section that provide an overview of the research discussed in that section.

2.2 Mathematical Optimization Models for the Fundamental Lifetime Maximization Problem

This section begins by formally describing and stating the fundamental lifetime maximization problem in Section 2.2.1. Because of its prevalence in the WSN optimization literature, we also describe a column-generation scheme for the solution of this problem in Section 2.2.2. Table 2.1 summarizes the most pertinent papers discussed in Section 2.2.

Author (Model)	Problem specification	Technique
General topology single-sink algorithms		
Chang and Tassiulas [1999] (FR and MREP)	Data-routing to balance energy consumption	LP formulation, two flow-augmenting algorithms that redirect flow
Xue et al. [2005]	Tree-based data-routing	$(1-\epsilon)$-approximation polynomial time algorithm to create a data-routing tree rooted at the sink
Madan and Lall [2004]	Partially-distributed data-routing	Lagrangian relaxation of an LP, subgradient algorithm to solve separable subproblems

Table 2.1: Section 2.2 summary

2.2.1 Fundamental Problem Description and Formulation

The fundamental problem we consider determines a data-routing plan consisting of a set of data flows from all targets to the sink in a manner that maximizes network lifetime. This problem can be formulated as a linear program (LP) introduced by Chang and Tassiulas [1999], who assume that each target is assigned *a priori* to a sensor. As such, data origination rates associated with the targets can instead be assigned directly to a sensor, allowing us to omit targets from our analysis. Consider a graph $\mathcal{G}$ having a set of nodes $\mathcal{N}$ composed of sensors $\mathcal{S}$ and a single sink. Let each sensor $i \in \mathcal{S}$ possess an initial battery power $P_i > 0$ and a total data origination rate of $b_i > 0$. The energy required by sensor i to send a unit of data to node j is $e_{ij} > 0,\ \forall i \in \mathcal{S},\ j \in \mathcal{N}$. Chang and Tassiulas assume that the energy required to receive a unit of data is negligible, but for the sake of generality, we define c_{hi} as the amount of energy required by sensor i to retrieve one unit of data from sensor h. Let variable z represent the WSN lifetime and variables y_{ij} be the data transmitted from node $i \in \mathcal{S}$

to $j \in \mathcal{N}$. The following LP maximizes WSN lifetime [Chang and Tassiulas, 1999].

$$\max \ z \tag{2.1a}$$

$$\text{s.t.} \ \sum_{j \in \mathcal{N}} y_{ij} - \sum_{h \in \mathcal{S}} y_{hi} - b_i z \ = 0 \quad \forall i \in \mathcal{S} \tag{2.1b}$$

$$\sum_{j \in \mathcal{N}} e_{ij} y_{ij} + \sum_{h \in \mathcal{S}} c_{hi} y_{hi} \leq P_i \quad \forall i \in \mathcal{S} \tag{2.1c}$$

$$y_{ij} \geq 0 \quad \forall i \in \mathcal{S}, \ j \in \mathcal{N} \tag{2.1d}$$

The objective function (2.1a) and constraints (2.1b) maximize WSN lifetime while ensuring flow balance. Constraints (2.1c) enforce energy-capacity constraints on sensor $i \in \mathcal{S}$ and constraints (2.1d) guarantee that no flows become negative.

Consider the following single-sink WSN lifetime maximization example in Figure 2.6, where a WSN contains sensors A, B, C, and D, and one sink, E. The edge labels denote the distance between each pair of objects. For this example, c- and e-values equal the distance between objects (e.g., a unit of flow from A to C consumes 10 units of energy at both A and C). We assume that sensors A and B are unable to transmit data directly to E. Sensors possess $P_i = 500$ units of energy, $\forall i = A, B, C, D$. Additionally, the amount of data originating at each sensor is $b_A = 8$, $b_B = 10$, $b_C = 0$, and $b_D = 0$ units/hour.

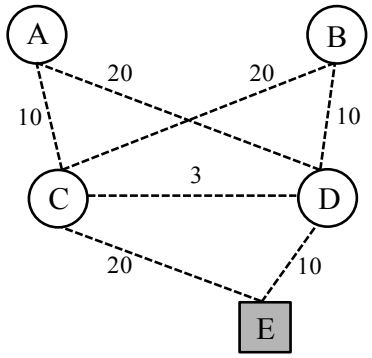


Figure 2.6: Single-sink WSN distances

We define the lifetime, L_i, of sensor i to be the number of hours that sensor i retrieves and transmits data given its battery power. Letting y_{ij} be the data flow per hour on arc (i, j), the energy required per hour by sensor $i \in \mathcal{S}$ is T_i. The lifetime for sensor i becomes $L_i = P_i / T_i$. We then define the WSN lifetime to be the minimum lifetime among all sensors. Figure 2.7 presents an optimal set

of data flows per hour that maximizes WSN lifetime.

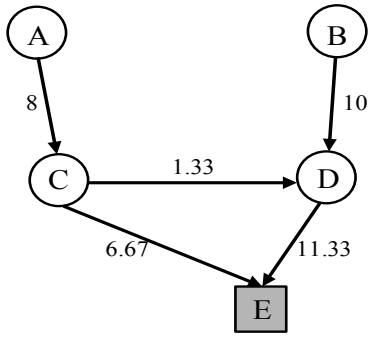


Figure 2.7: Optimal data flows per hour

Given these flows, the lifetime of sensor A is $500/80 \approx 6.3$ hours, sensor B is $500/100 = 5$ hours and sensors C and D are $L_C = L_D = 500/217.3 \approx 2.3$ hours. Therefore, the maximum WSN lifetime is 2.3 hours.

2.2.2 Column Generation Approach

For a fixed value of z, the formulation given by (2.1) contains traditional network flow constraints (2.1b), which are then complicated by the presence of energy-capacity constraints (2.1c). The latter set of constraints are accordingly referred to as *complicating constraints* in the literature. In general, complicating constraints are those that, when removed, reveal a constraint structure that is more amenable to solution via mathematical optimization or specialized algorithms, thus reducing solution time and possibly decentralizing decision-making [Dantzig and Wolfe, 1960]. Two notable decomposition methods that exploit this special structure are column generation (CG) [Desrosiers and Lübbecke, 2005] and subgradient optimization of a Lagrangian relaxation (LR) problem [Fisher, 1985].

We focus on describing a CG approach for solving (2.1); see also [Alfieri et al., 2007, Behdani et al., 2012, Castaño et al., 2015, 2014, Castao et al., 2013, Gentili and Raiconi, 2013, Rossi et al., 2012b,a, Türkoğullari et al., 2010]. For an example of LR applied to (2.1), see [Madan and Lall, 2004]. Let $\mathcal{K}$ be the set of all possible data-routing patterns, where a pattern is given by an acyclic set of flows on $\mathcal{G}$ that satisfies (2.1b) and (2.1d) with z fixed to 1. Define constants y_{ij}^k as the amount of flow from sensor $i \in \mathcal{S}$ to $j \in \mathcal{N}$ in pattern $k \in \mathcal{K}$. The following LP contains variables λ_k that

represent the amount of time we send flow on pattern $k \in \mathcal{K}$, and is equivalent to model (2.1).

$$\max \sum_{k \in \mathcal{K}} \lambda_k \tag{2.2a}$$

$$\text{s.t.} \sum_{k \in \mathcal{K}} \left(\sum_{j \in \mathcal{N}} e_{ij} y_{ij}^k + \sum_{h \in \mathcal{S}} c_{hi} y_{hi}^k \right) \lambda_k \leq P_i \quad \forall i \in \mathcal{S} \qquad (\alpha_i) \tag{2.2b}$$

$$\lambda_k \geq 0 \quad \forall k \in \mathcal{K} \tag{2.2c}$$

The objective function (2.2a) maximizes WSN lifetime while constraints (2.2b) enforce energy-capacity constraints on each sensor and constraints (2.2c) enforce flow nonnegativity.

By enumerating all possible patterns, model (2.2) is equivalent to (2.1) [Desrosiers and Lübbecke, 2005]. Since a majority of patterns in $\mathcal{K}$ will be non-basic at an optimal solution to (2.2), we formulate a restricted master problem (RMP) that maximizes network lifetime considering only a subset of all patterns $\mathcal{K}^* \subset \mathcal{K}$. Suppose that we optimize this RMP and identify primal and dual optimal solutions. (The RMP must have a feasible solution since the trivial solution $\lambda_k = 0,\ \forall k \in \mathcal{K}^*$ is feasible. The fact that all e-values are positive guarantees that the RMP is bounded. These facts guarantee the existence of an optimal solution to the RMP.) Let α_i be the optimal dual values for the energy-capacity constraints (2.2b) in the RMP. We formulate an LP subproblem to find a new data-routing pattern having a positive reduced cost. Letting the continuous variable w_{ij} be the flow from sensor $i \in \mathcal{S}$ to $j \in \mathcal{N}$, we formulate the following subproblem:

$$\min \sum_{i \in \mathcal{S}} \alpha_i \left(\sum_{h \in \mathcal{S}} c_{hi} w_{hi} + \sum_{j \in \mathcal{N}} e_{ij} w_{ij} \right) - 1 \tag{2.3a}$$

$$\text{s.t.} \sum_{j \in \mathcal{N}} w_{ij} - \sum_{h \in \mathcal{S}} w_{hi} - b_i = 0 \quad \forall i \in \mathcal{S} \tag{2.3b}$$

$$w_{ij} \geq 0 \quad \forall i \in \mathcal{S},\ j \in \mathcal{N}, \tag{2.3c}$$

where the objective function (2.3a) is the negative of the reduced cost of a data-routing pattern. This negation is performed to show that the subproblem (also called the *pricing problem*) reduces to a min-cost-flow problem with nonnegative costs.

The CG algorithm proceeds as follows. The RMP is solved with some set of initial feasible patterns in $\mathcal{K}^*$. We obtain the dual values α_i to formulate the objective function of (2.3). If the

optimal objective function value to (2.3) is negative, then the identified pattern has a positive reduced cost. We add that pattern to $\mathcal{K}^*$ and re-solve the RMP. This process repeats until no positive reduced-cost patterns are found, which implies that the current solution to the RMP is also optimal to (2.1).

For large networks, solving (2.1) directly or by a decomposition method may require significant computational effort. Thus, Chang and Tassiulas [1999] present two flow-augmenting heuristics: the flow redirection (FR) and minimum residual energy path (MREP) algorithms. Starting with an initial feasible data-routing plan, both algorithms seek to balance flow among all target-to-sink paths. The balancing strategy establishes a mechanism for evaluating path length, and the algorithms redirect a portion of flow from a longest path to a shorter path. The FR algorithm defines path length as the minimum lifetime of sensors on the path, whereas the MREP defines it as the minimum inverse residual energy of sensors along the path. The FR algorithm produces solutions whose objectives are on average 95% of optimality, while the MREP algorithm produces a ratio of 96%.

Xue et al. [2005] create an approximation algorithm to the maximum lifetime problem as opposed to solving (2.1). Their algorithm constructs a routing solution based on path augmentation, where arc lengths are a function of the data currently transmitted on them. Based on those lengths, the authors iteratively employ the Garg and Köneman [1998] to find a shortest path from all sensors to the sink, push a limited amount of flow on the shortest among these paths, and then recalculate the arc lengths. The authors demonstrate that their polynomial-time algorithm provides a $(1 - \epsilon)$-approximation, for any given parameter $\epsilon > 0$.

The elegant LP (2.1) and associated algorithms mentioned in this section are effective for many classes of WSN optimization problems, but even minor adjustments to the assumptions on how these networks function necessitate completely different algorithms for these problems. Moreover, many other considerations exist beyond data-routing and energy utilization in these networks. The following sections explore these models and algorithms in more detail.

2.3 Extensions to the Fundamental Lifetime Maximization Problem

In Section 2.3 we examine several extensions to the fundamental lifetime maximization problem presented in Section 2.2. First, Section 2.3.1 details online algorithms, which determine data

transmission paths without knowledge of future data-routing requests. In Section 2.3.2, we examine the case when sensors are partitioned into multiple clusters to improve scalability. In Section 2.3.3, we consider the case in which sensors are uniformly deployed to form a grid-like topology. Then, in Section 2.3.4, we review protocols and algorithms used to form virtual tree backbones among sensors in the network. Tables 2.2 and 2.3 summarize key papers found in Sections 2.3.1 and 2.3.2, respectively, and Table 2.4 summarizes those found in Sections 2.3.3 and 2.3.4.

Author (Model)	Problem specification	Technique
	Online algorithms	
Aslam et al. [2003]	Zone-based data-routing	Avoid high total-energy paths, maximize the smallest residual-energy fractional edge in a path
Toh et al. [2011]	Data-routing	Avoid low-remaining-energy sensors, minimize total energy consumed on a path
Park and Sahni [2006] (OML)	Distributed data-routing	Successive minimum weight path routing, where weight is a function of residual energy
Mohanoor et al. [2009]	Data-routing	Minimum weight routing as a function of residual energy, tie-breaking by energy consumption
Wahid et al. [2014]	Distributed data-routing on dynamic underwater networks	Myopic routing based on distance to sink and residual energy

Table 2.2: Section 2.3.1 summary

2.3.1 Online Algorithms

Solution techniques to the problem presented in Section 2.2 are said to be *offline* since all data-routing requests are known before the routes are planned. Alternatively, *online* data-routing algorithms consider the case in which data-routing requests arrive nonsimultaneously. These algorithms must therefore determine data transmission paths without knowledge of future requests. As such, the optimal maximum lifetime assuming offline routing is an upper bound on the maximum lifetime assuming online routing.

Aslam et al. [2003] propose an online heuristic that tends to avoid data transmission through low-energy sensors while choosing relatively low total-energy paths. Define the residual energy

Author (Model)	Problem specification	Technique
Clustering algorithms		
Heinzelman et al. [2000] (LEACH)	Distributed clusterhead selection	Localized clusterhead selection based on residual energy levels of neighboring sensors
Heinzelman et al. [2002] (LEACH-C)	Clusterhead selection	Simulated annealing approach to cluster construction and clusterhead selection
Kumar et al. [2009] (EEHC)	Clusterhead selection	Restrict sensors that cover targets, avoid selecting low-energy sensors as clusterheads
Muruganathan et al. [2005]	Clusterhead selection and data-routing	Clusters chosen based on residual energy, minimum spanning tree for routing and data scheduling
Javaid et al. [2013]	Dynamic clusterhead selection	Periodic clusterhead selection based on residual energy levels
Ducrocq et al. [2013]	Distributed clusterhead selection	Clusterhead selection based on node degree, density, and residual energy, tree construction for routing
Leu et al. [2015] (REAC-IN)	Distributed zone-based clusterhead selection	Based on average residual energy and distance between sensors within each zone
Nikolidakis et al. [2013]	Clusterhead selection and scheduling	LP to minimize clusterhead energy consumption, scheduling protocol to avoid data collisions
Latiff et al. [2007] (PSO-C)	Clusterhead selection and data-routing	PSO algorithm based on residual energy levels, scheduling protocol to avoid data collisions
Singh and Lobiyal [2012]	Clusterhead selection with data retransmission	PSO algorithm based on intra-cluster distances
Kuila and Jana [2014]	Cluster formation and data-routing	PSO algorithm to solve optimization models that balance energy usage among intra-cluster sensors

Table 2.3: Section 2.3.2 summary

fraction of edge (i, j) as $u_{ij} = (P_i - e_{ij})/P_i$. The heuristic evaluates the quality of a path by the smallest residual energy fraction edge that lies on the path. The goal is to find the highest quality path possible, subject to the restriction that the total energy consumed by the path cannot exceed some maximum threshold value, τ. Their algorithm first employs Dijkstra's algorithm to find a

minimum energy path and computes the minimum residual energy fraction u_{min} among all edges on the path. The algorithm then removes all edges whose residual energy fraction is no more than u_{min} from the graph, and finds a new minimum energy path. When the energy consumed by a minimum energy path exceeds τ, or when no path exists, the data-routing request is transmitted on the minimum energy path from the most recent iteration. This approach requires centralized coordination, which may be difficult to implement for large networks. Thus, the authors also propose a hierarchical zone-based approach to implement their algorithm within smaller sensor zones. The authors conclude that their initial approach achieves over 90% of the optimal offline data-routing objective for many instances.

Whereas Aslam et al. [2003] place an upper bound on a path's energy consumption, Toh et al. [2011] present an algorithm that minimizes the energy consumption of each data transmission path, in which data may only be transmitted through sensors having remaining energy levels above some threshold. Their approach trims the network of low-energy sensors and implements a shortest-path algorithm to determine each path, where edge weights are defined as the energy required to transmit data between two nodes. If no path exists in the trimmed network, then their algorithm chooses a path in the original network that maximizes the minimum remaining energy level among all sensors on the path.

Park and Sahni [2006] also present an online maximum lifetime (OML) heuristic, in which they define the residual energy of edge (i, j) to be $P_i - e_{ij}$. Their algorithm first trims the network of all edges requiring more energy than available for transmission, determines a minimum energy path in the resulting network, computes the minimum residual energy ($minRE$) of all edges on this path, and finally removes all edges whose residual energy is less than $minRE$. The authors then assign a weight to each remaining edge as a function of e_{ij} and E_i. This weighting function assigns a relatively high weight to each edge whose use will reduce a sensor's energy level below some minimum threshold. The authors then choose a minimum-weight path on which the data-routing request is transmitted. The authors also present a distributed version of the OML algorithm that divides sensors into clusters, with each cluster having a designated clusterhead (see Section 2.3.2 for more research on WSN clustering). The OML algorithm results in higher average lifetime values than those found in [Aslam et al., 2003] and [Toh et al., 2011].

Mohanoor et al. [2009] extend the previous works to develop a two-phase online algorithm that maximizes lifetime if multiple shortest paths exist in the trimmed network for each incoming

data-routing request. In the first phase, a variant of Dijkstra's algorithm computes a path that maximizes the minimum residual energy, B, among all sensors on the path, where residual energy is again defined as in [Park and Sahni, 2006]. If multiple shortest paths are found during phase one, the second phase removes edges whose residual energy is less than B and selects (from among the shortest paths in the first phase) a path that minimizes total energy consumption. The authors show that their algorithm prolongs WSN lifetime compared to algorithms in [Aslam et al., 2003] and [Park and Sahni, 2006].

Online routing algorithms are especially useful in applications where network information dynamically changes due to environmental influences. One such application arises in underwater WSNs, where sensors may be periodically relocated due to ocean currents. Wahid et al. [2014] formulate an online routing protocol to maximize the number of successful data transmissions for underwater WSNs. Rather than individually routing each incoming request, their protocol determines a data-routing plan for a set of requests at each sensor over some time period. To determine this plan, their protocol first computes edge lengths and quality by transmitting minuscule messages between neighboring sensors. Each sensor then transmits data on a high-quality edge to a node closer to the sink having sufficient residual energy. This process iterates at each node until the data transmission arrives at the sink. Since currents constantly change sensor positions, their protocol intermittently updates edge length, edge quality, and residual sensor energy levels, and determines a new data-routing plan for the next set of incoming requests. Their approach outperforms distance-based routing protocols for underwater WSNs in terms of network lifetime, energy consumption, and transmission delay.

2.3.2 Clustering Techniques

Sensors can be partitioned into multiple clusters to employ techniques that reduce the size of the network and simplify data-routing decisions. These techniques divide sensors into clusters, each having a designated clusterhead that computes local routing decisions among sensors within its own cluster (Figure 2.8). Creating sensor clusters also minimizes the burden of communication since only the clusterhead communicates with the sink. Some studies assume that clusters are given (e.g., [Latiff et al., 2007]), while others examine the generation of the clusters (e.g., [Kuila and Jana, 2014, Singh and Lobiyal, 2012]). Other research strategies use sensor clustering to decentralize network protocols [Javaid et al., 2013, Liu and Cao, 2012, Nikolidakis et al., 2013, Singh and Lobiyal, 2012].

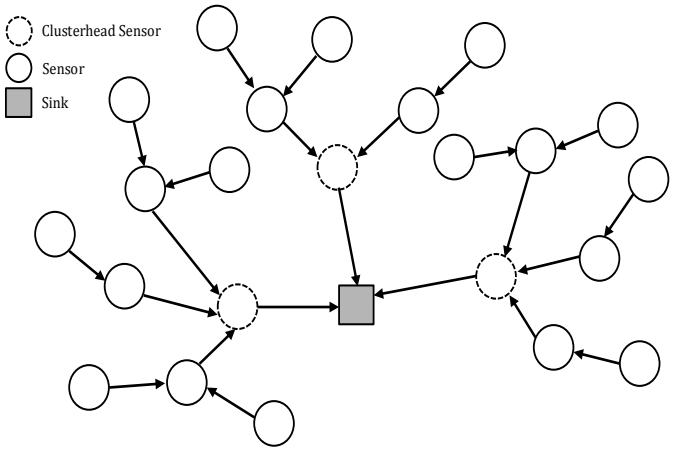


Figure 2.8: WSN clustering method

In addition to data-routing decisions, clustering algorithms often need to determine which node will serve as the clusterhead. Heinzelman et al. [2000] present the low-energy adaptive clustering hierarchy (LEACH) protocol, which uses localized coordination and control for cluster formation and operation. In this protocol sensors elect themselves to serve as the clusterhead based on residual energy levels among neighboring sensors. Sensors then transmit data directly to the most favorable clusterhead to form clusters. The clusterhead creates a schedule of data-flow paths from each sensor in its cluster to avoid data collisions, which may occur when a clusterhead simultaneously receives data from multiple sensors, thus resulting in inaccurate data transmissions. To balance energy consumption, clusterhead locations are rotated according to some probability when the current clusterhead's residual energy reaches a minimum threshold. Maximum lifetime values found using LEACH were significantly larger than those computed by algorithms that primarily focus on minimizing energy consumption.

Sensors employing the LEACH protocol rely on localized information to elect themselves as clusterheads independent of energy levels at non-neighboring sensors, which can lead to uneven distribution of clusterheads within the WSN. To ameliorate this problem, Heinzelman et al. [2002] formulate the centralized LEACH (LEACH-C) protocol in which each sensor broadcasts its information across the network. Considering only sensors having above-average energy levels, LEACH-C implements a simulated annealing algorithm to find a near-optimal set of k clusters and clusterheads. The energy efficient heterogeneous clustered scheme (EEHC) by Kumar et al. [2009] provides a related approach for the case in which some predetermined set of sensors is only allowed to monitor

targets, while sensors having high initial energy levels can relay, process, and transmit data to the sink.

Muruganathan et al. [2005] also implement a centralized clustering protocol to balance energy consumption in which only sensors having above-average residual energy levels serve as the clusterhead. At each iteration of the algorithm, some central node determines a set of clusterheads and partitions sensors into clusters to balance the load of each clusterhead. (One cluster of sensors will have the sink represent the clusterhead.) For inter- and intra-cluster routing decisions, their protocol creates a minimum spanning tree rooted at the clusterhead to minimize energy consumption and ensure full connectivity. They additionally provide a scheduling scheme to minimize the number of data collisions. Their approach ensures that each clusterhead serves approximately the same number of sensors to avoid early exhaustion and ensure an even distribution of clusterheads throughout the network. As a result, their approach reduces overall energy consumption and prolongs network lifetime when compared to LEACH and LEACH-C.

The previous approaches are designed for static sensing environments. For the dynamic case, the choice of clusterhead should change depending on evolving sensing needs. Accordingly, Javaid et al. [2013] examine a dynamic approach to the clusterhead selection problem. After a sensor serves as the clusterhead for some predetermined period of time, their approach updates residual energy levels and chooses a sensor having the maximum residual energy level to serve as the clusterhead. This technique enables dynamic adaptation to the environment, allowing data flows to balance energy consumption across the network. Simulations show that this approach results in longer lifetime values than static clustering protocols.

For WSNs incapable of centralized sensor coordination, Ducrocq et al. [2013] propose an energy-aware clustering protocol to create non-overlapping clusters in a distributed fashion. This protocol considers node degree and density in addition to residual sensor energy levels when selecting a clusterhead. Each link is initially assigned a weight as a function of the residual energy of the nodes incident to the link, where a large weight corresponds to an edge connecting high-energy sensors. In the first step, their protocol trims the WSN graph by removing an edge having the smallest weight in every triangle of the graph. Once the graph is reduced, some predetermined number of clusterheads are selected based on residual energy levels, and clusters are formed through a tree construction algorithm. For a semi-distributed implementation, each sensor broadcasts its residual energy, node degree, and density level to neighboring sensors. Their algorithm provides longer network lifetime

values than LEACH for WSNs having relatively few clusterheads, while LEACH performs better in WSNs having a large number of clusterheads.

Unbalanced cluster formations can also lead to isolated sensors, defined as any sensor that is located excessively far from the nearest clusterhead, and thus requires substantial energy expenditures to transmit data to a neighboring sensor. To minimize energy consumed at such a node, Leu et al. [2015] propose the regional energy aware clustering with isolated nodes (REAC-IN) protocol to select clusterheads based on the residual energy of each sensor, the average residual energy of sensors within a cluster, and the distance between each sensor and proposed clusterhead. Their protocol identifies isolated nodes and decides whether data at such nodes should be transmitted to the nearest clusterhead or directly to the sink, depending on the average residual energy among neighboring sensors and the distance between those sensors and the sink.

Rather than taking a heuristic approach, Nikolidakis et al. [2013] propose a centralized LP model for the cluster formation and clusterhead selection problems to minimize the energy consumption at each clusterhead. A solution to this problem provides the frequency and duration of time each sensor serves as a clusterhead. After solving the LP, they implement a scheduling protocol to avoid data collisions. Furthermore, the authors solve their LP again to determine a new set of clusterhead locations if there is some change in node position or energy consumption. Results show that their approach not only minimizes energy consumption but maximizes network lifetime compared to LEACH [Heinzelman et al., 2000] and the algorithms presented by Muruganathan et al. [2005].

Another portion of the literature employs particle swarm optimization (PSO) for clustering decisions related to WSN lifetime maximization. A PSO algorithm consists of a swarm or pool of candidate solutions called particles that explore some solution space in search of a global optimal solution [Kennedy, 2011]. Each particle possesses a fitness function to evaluate the quality of the associated solution. Particles also use velocity and direction vectors to iteratively move through the search space until either an acceptable solution is found or a fixed number of iterations is reached.

Latiff et al. [2007] implement a centralized energy-aware clustering PSO algorithm (PSO-C) for the clusterhead selection and data-routing problems. During the setup phase of PSO-C, each sensor notifies the sink of its position and residual energy level to compute the average residual energy among all sensors. Clusterheads may only be selected from among sensors having above-average residual energy levels. To finish the setup phase, the PSO-C selects the k best sensors to serve as

clusterheads, based on a particle fitness function that rewards minimum intra-cluster distances and balanced energy consumption. The authors then employ a scheduling protocol to coordinate data transmissions with the goal of avoiding data collisions. This two-phase algorithm iterates until an acceptable solution is found. Their algorithm evenly distributes clusterheads throughout the network to provide higher network lifetime values than those given by LEACH [Heinzelman et al., 2000] and LEACH-C [Heinzelman et al., 2002] algorithms.

Singh and Lobiyal [2012] also take a PSO approach for generating energy-aware clusters. Their approach seeks to minimize the average distance between a sensor and its clusterhead, using a particle fitness function that combines average transmission distance, residual energy levels, node degree, retransmission levels, and the number of times a sensor serves as the clusterhead. Rather than preventing data collisions, their approach simply retransmits initially unsuccessful transmissions. They seek to minimize data retransmissions by avoiding sensors prone to data collisions. The authors show that their algorithm achieves better energy savings and WSN lifetime values than those produced by PSO-C [Latiff et al., 2007].

Kuila and Jana [2014] present LP and nonlinear programming (NLP) models for the WSN data-routing and cluster formation problems, respectively. The routing approach is an LP that jointly minimizes the maximum data transmission distance and the maximum number of transmission hops. The objective function for the NLP balances energy consumption by maximizing the ratio of the minimum clusterhead lifetime to the average distance between sensors and their clusterhead. Due to the difficulty of solving these two models, the authors propose PSO-based algorithms for their solution. For routing decisions, the particle fitness function is based on the objective function of their LP model. In the clustering problem, the fitness function is based on energy balance and conservation among sensors in each cluster. The authors show that the proposed algorithms result in better network lifetime and total transmitted data than existing algorithms.

2.3.3 Grid-based Topology

In some WSN application areas, sensors are deployed in a grid-like topology (Figure 2.9) in which sensors are uniformly deployed [Saad and Tourancheau, 2009]. Such topologies balance sensor distribution and simplify data-routing decisions. Kacimi et al. [2013] formalize the WSN lifetime maximization problem for two-dimensional grid-based network topologies to derive an optimal load-balanced data-routing plan. They formulate an NLP model to minimize the maximum energy

consumption among all sensors, and solve this NLP via a heuristic scheme. Using this heuristic, a sensor distributes data among multiple shortest paths according to the residual energy levels at each forwarding sensor.

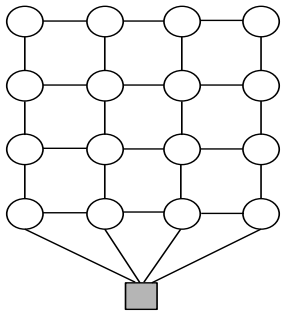


Figure 2.9: Square grid topology

In addition to routing decisions, Li et al. [2014] consider two cases of sensor deployment to further reduce energy consumption in grid-based WSNs. Assuming the sink is located at the center of the sensing area, the authors consider partitioning this area into hexagonal cells (Figure 2.10). In the uniform sensor deployment problem, a single sensor may be placed at the center of each cell, and in the non-uniform version, a variable number of sensors may be be placed in cells nearest the sink. Due to restrictions on transmission range, sensors are only allowed to transmit data to sensors in neighboring cells. Given this grid-based topology, the authors seek to determine an optimal uniform distance between all neighboring sensors that ensures full coverage and balances energy consumption. The authors formulate an NLP model for each case of sensor deployment and employ a greedy protocol for all data transmission decisions, in which data is transmitted to the sensor closest to the sink. The authors employ a generalized reduced gradient method [Gabriele and Ragsdell, 1977] to solve the NLP for uniform sensor deployment and a genetic algorithm (GA) for the non-uniform sensor deployment case.

Zhang et al. [2016] propose two grid-based topology control algorithms to minimize the number of active sensors required to maximize WSN lifetime. The algorithm initially identifies the midpoint of the sensing area and divides it into equally-sized square cells that may or may not contain sensors. Additionally, the authors designate a sensor (if one exists) within each cell to serve as the clusterhead and adds edges between all pairs of clusterheads located in adjacent cells. Their algorithm then builds a near-minimum connected dominating set (CDS) for this new clusterhead graph, where a CDS is a subset of clusterheads such that a path exists between every pair of clusterheads in the

Author (Model)	Problem specification	Technique
	Grid-based topology algorithms	
Kacimi et al. [2013]	Grid-based data-routing	Heuristic solution to NLP that distributes data among multiple low-energy paths
Li et al. [2012]	Grid-based sensor deployment and data-routing	Solve NLP to determine hexagon-based sensor density, provide a greedy protocol for routing
Zhang et al. [2016]	Grid-based sensor deployment and clusterhead selection	Near-minimum CDS-based algorithm for clusterhead selection
	Virtual backbone techniques	
He et al. [2013]	Virtual backbone formation and non-backbone allocation	NLP used to determine a load-balanced backbone, IP used to assign non-backbone sensors to backbone
Zhao et al. [2011]	Virtual backbone formation and scheduling	Avoid selecting low-energy sensors for the backbone, graph-based algorithm for backbone scheduling
Rizvi et al. [2012]	Distributed virtual backbone formation	Depth-first search to minimize energy consumed when forming backbone
Luo et al. [2006]	Tree-based data fusion and routing	Minimum cost perfect matching algorithm to construct fusion and routing tree

Table 2.4: Section 2.3.3 and 2.3.4 summary

CDS, and all clusterheads not in the CDS are adjacent to a clusterhead in the CDS. Furthermore, the authors propose an algorithm to determine the minimum number of columns or rows in the grid needed to cover all network sensors and form a CDS.

2.3.4 Virtual Backbone Techniques

Another line of research considers forming a virtual backbone within a WSN (Figure 2.11) to ensure full coverage and connectivity regardless of sensor deployment. A virtual backbone is composed of a subset of sensors tasked with forwarding data retrieved by neighboring sensors. The backbone nodes, along with the sink, establish a CDS in the network. A routing tree structure is typically formed over these nodes, rooted at the sink. Routing is performed by transmitting data

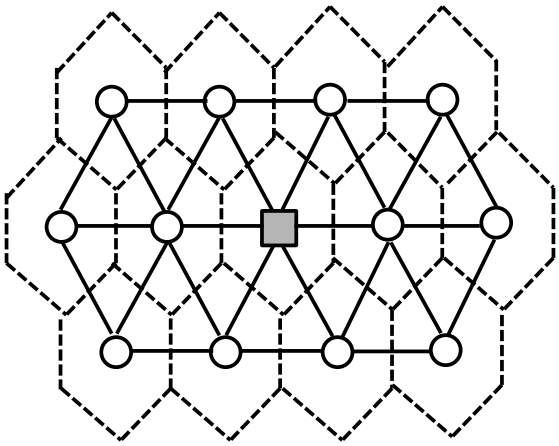


Figure 2.10: Hexagon grid topology

from each non-backbone sensor to the most favorable backbone sensor, which transmits that data through the backbone tree to the sink.

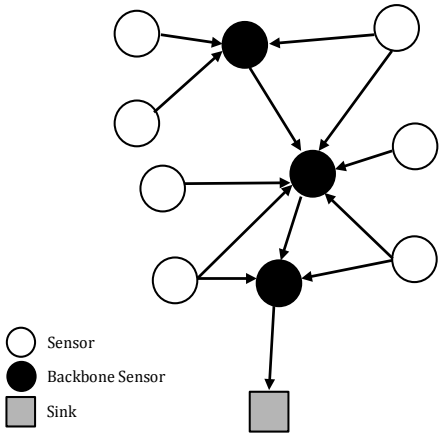
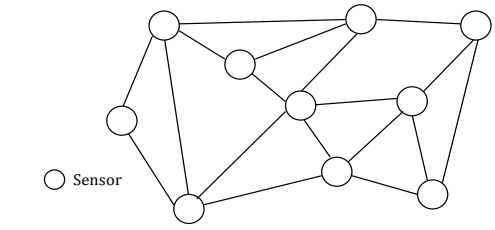


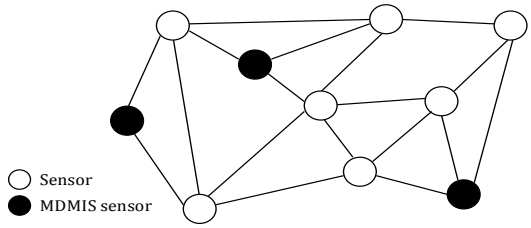
Figure 2.11: WSN backbone

He et al. [2013] investigate a three-phase approach to establish a backbone that balances energy consumption among all sensors. The first phase considers the min-max degree maximal independent set (MDMIS) problem, which seeks to find a maximal independent set that minimizes the maximum degree of any sensor in the set. The authors formulate an integer NLP for the MDMIS problem and use an approximation algorithm that solves the linear programming relaxation and rounds any fractional values. (See Figure 2.12b for an example solution to the MDMIS problem.) The solution to this problem yields a subset of the backbone nodes. The second phase solves the load-balanced virtual backbone (LBVB) problem to identify a backbone that balances the number of

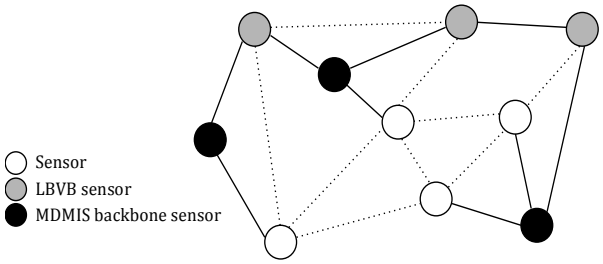
neighboring non-backbone sensors of each backbone sensor. The authors present a heuristic scheme to select a minimum set of additional nodes that, when connected to the nodes in the MDMIS solution, provide a solution to the LBVB problem. Using the LBVB solution, the third phase solves the min-max valid-degree non-backbone node allocation (MVBA) problem to assign each non-backbone sensor to a backbone sensor in a way that balances the transmission load of all backbone sensors (Figure 2.12c). The authors formulate the MVBA problem as an integer program (IP) and present an approximation algorithm based on a random rounding heuristic. The authors show that their three-phase approach can prolong WSN lifetime by up to 69% compared to other CDS-based virtual backbone algorithms [Zhao et al., 2011].



(a) Original WSN



(b) Backbone sensors after solving MDMIS



(c) Backbone after connecting MDMIS and LBVB sensors. Non-backbone sensors are connected to the backbone by solid lines.

Figure 2.12: Load-balanced virtual backbone

Rather than forming a single WSN backbone, Zhao et al. [2011] examine the formation of multiple backbones in dense WSNs. Their protocol alternates the sequence in which these backbones are used in order to balance energy consumption. The authors formulate the maximum lifetime

backbone scheduling (MLBS) problem to determine the schedule in which each backbone is used to retrieve and transmit data. They propose two approximation algorithms for the MLBS problem, which are based on a schedule transition graph (STG) and virtual scheduling graph (VSG). The STG-based algorithm constructs and schedules a polynomial set of n virtual backbones $bb_1, \ldots, bb_n$. Their algorithm maps this set of backbones to a transition graph (Figure 2.13), where the series of transition levels $T_1, \ldots, T_i$ corresponds to the backbone schedule sequence, in which each backbone transmits data for some predetermined period of time. The authors define the energy level of the network after transmitting data on backbone bb_j at transition level T_k to be a tuple of the residual energy values of all sensors. The STG-based algorithm is as follows. After the network transmits data on some backbone at T_k, the authors determine the set of n energy levels associated with the case when each backbone $bb_1, \ldots, bb_n$ is implemented at T_{k+1}. From among this set, they select the backbone resulting in an energy level with the maximal minimum residual energy value to be utilized at T_{k+1}. After the network transmits data on this backbone, their algorithm iterates until a sensor exhausts its energy to determine a backbone schedule.

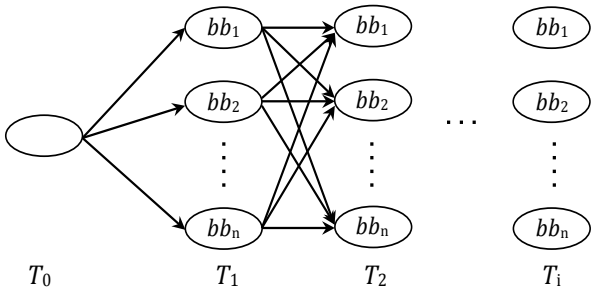
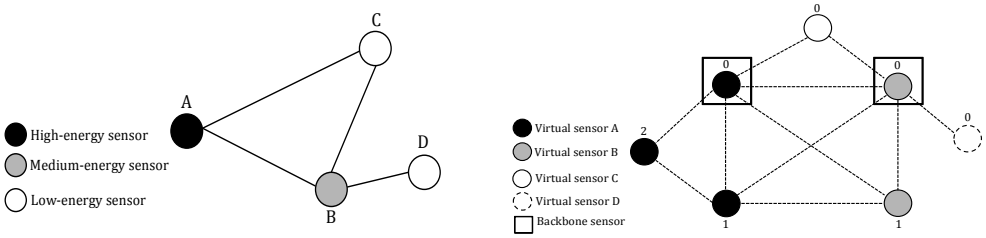


Figure 2.13: Schedule transition graph

For the VSG-based algorithm, each sensor i in the original WSN is transformed into a clique of E_i virtual nodes, where E_i is a positive integer that corresponds to the remaining energy at sensor i. The virtual nodes are indexed from $0, \ldots, E_i - 1$. An edge connects the uth virtual node for sensor i with the vth virtual node for sensor j if and only if (a) an edge connects sensors i and j in the original WSN, and (b) $\max\{u, v\} \leq \min\{E_i, E_j\} - 1$. Figure 2.14 illustrates this transformation. In particular, note that $E_A = 3$ and $E_B = 2$, and so there exist edges between virtual node u associated with sensor A and both virtual nodes associated with sensor B, for $u = 0$ and 1. However, virtual node 2 associated with sensor A is not connected to virtual nodes associated with sensors B or C because E_B and E_C are both less than 3.

By forming the virtual graph in this way, sensors having high energy tend to be a part of the backbone. The authors then obtain a minimum CDS in the virtual graph that does not contain multiple virtual nodes of the same sensor. Finally, the authors remove all duplicate virtual nodes to reveal a backbone in the original graph. They also present a distributed algorithm for the MLBS problem, in which each backbone sensor finds a replacement node among neighboring sensors to form a new backbone.



(a) Original WSN

(b) Virtual graph

Figure 2.14: Virtual scheduling graph, in which $E_A = 3$, $E_B = 2$, $E_C = 1$, and $E_D = 1$

Rizvi et al. [2012] seek to find a set of sensors that form a CDS corresponding to a virtual backbone to reduce the size of the network and simplify routing decisions. The authors define active sensors as those sensors able to relay data through the network, and non-active sensors as those that only retrieve data. The authors present a tree-based distributed algorithm that starts by forming a virtual graph containing all sensors using a depth-first search from some initial sensor. Each leaf node of this graph is designated as non-active sensor with the other sensors being active. The authors conclude that their algorithm consumes less energy when forming a virtual backbone compared to other CDS-based algorithms.

Network trees are useful in WSNs with data fusion capabilities. Data fusion is the process of locally aggregating information retrieved from multiple sensors before being transmitted to another node [Chen et al., 2004, Luo et al., 2006]. Luo et al. [2006] implement a routing algorithm to determine an energy efficient data-routing plan in WSNs implementing data fusion. The authors first propose an IP to solve this problem, in which integer variables determine if data is fused at a sensor. Due to the difficulty of solving this model, they present an approximation algorithm that first employs a shortest-path algorithm to find a minimum-cost path between every pair of non-sink nodes (u, v), where edge costs are a function of the energy required for transmission from u to v and the energy required for fusion at v. Their algorithm then finds a minimum-cost perfect matching

between all non-sink nodes to construct a data routing and fusion tree that minimizes the total energy consumed. Their algorithm achieves a tree with a $\frac{5}{4}\log(n+1)$ approximation to the optimal objective, where n is the number of sensors in the network.

2.4 Extensions Based on Sink Characteristics

We now describe various extensions to the lifetime maximization problem based on sink properties. In Section 2.4.1 we explore lifetime maximization algorithms for WSNs having multiple sinks. We then present in Section 2.4.2 extensions to model (2.1) that consider the presence of a mobile sink. Finally, we examine the lifetime maximization problem for WSNs having multiple mobile sinks in Section 2.4.3. Tables 2.5 and 2.6 summarize key papers in Sections 2.4.1 and 2.4.2, respectively, and Table 2.7 summarizes those in Section 2.4.3.

Author (Model)	Problem specification	Technique
	Multi-sink models	
Xue et al. [2005]	Tree-based data-routing	$(1-\epsilon)$-approximation polynomial time algorithm to create a set of trees, each rooted at a sink
Castao et al. [2013]	Data-routing and scheduling	CG scheme to maximize lifetime, greedy randomized algorithm to solve subproblem
Kim et al. [2005]	Constrained sensor and sink deployment and data-routing	MIP to determine sink locations and data-flows
Türkoğullari et al. [2010]	Sensor and sink deployment and data-routing	Require a sufficiently large number of sinks be within a minimum distance to sink, CG scheme for routing

Table 2.5: Section 2.4.1 summary

2.4.1 Multiple-Sink WSNs

Introducing multiple sinks can extend WSN lifetime by improving energy consumption balance across the network [Mansouri et al., 2008]. Model (2.1) can be extended to address the multi-sink WSN lifetime maximization problem, assuming either that collected data can be transmitted to any sink, or that each data-routing request has a specific sink to which it is transmitted. In this

Author (Model)	Problem specification	Technique
Non-delay tolerant single mobile-sink models		
Papadimitriou and Georgiadis [2006] (MSM)	Sink traversal and data-routing	LP for sink movement and dwelling times and routing
Wang et al. [2005]	Sink traversal on a uniform two-dimensional grid	LP for sink traversal, shortest-path algorithm for routing
Huang et al. [2015]	Reactive sink relocation on a hexagonal grid	Sink moves to nearby area with higher average residual energy, shortest-path algorithm for routing
Gatzianas and Georgiadis [2008]	Distributed sink traversal and routing	Subgradient algorithm for Lagrangian relaxation of LP, minimum-cost flow algorithm for routing
Liang and Luo [2011]	Distance-constrained sink tour and hop-constrained routing	Heuristic that determines sink trajectory first, then dwelling times at each location
Delay-tolerant single mobile-sink models		
Yun and Xia [2010] (DT-MSM)	Sink traversal and routing varying delay restrictions	LP to determine sink traversal and dwelling times
Yun et al. [2010a]	Distributed sink traversal and routing	Lagrangian relaxation of an LP, fractional knapsack algorithm for subproblem at each sensor
Behdani et al. [2012]	Partially-distributed sink traversal and routing	CG scheme with separable shortest-path subproblems at each sensor
Keskin et al. [2011]	Sink traversal and routing with non-negligible sink travel time	Heuristic for an MIP with an upper bound on the number of transmission hops
Behdani et al. [2013]	Sink traversal and routing with non-negligible sink travel time	Branch-and-cut approach to solving an MIP with subtour elimination constraints

Table 2.6: Section 2.4.2 summary, where sink travel times are negligible unless otherwise specified

chapter, we employ the former assumption.

We extend the prior single-sink WSN example to demonstrate how the presence of a second sink prolongs network lifetime. Figure 2.15 presents a two-sink example similar to the WSN example in Figure 2.6.

A new sink E_1 is added to the network while sink E_2 remains at the same location as E in the single-sink example. The energy required to send a unit of data from sensors C and D to E_1 is

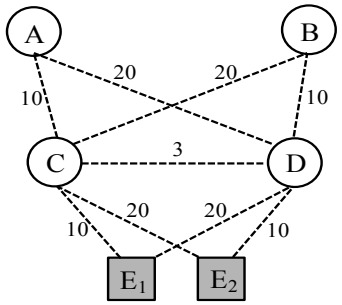


Figure 2.15: Two-sink WSN example

$e_{CE_1} = 10$ and $e_{DE_1} = 20$ units. All other information regarding battery power, data origination amounts, and energy expenditures are the same as those used in the single-sink WSN example. Figure 2.16 presents an optimal set of data flows per hour for the two-sink WSN.

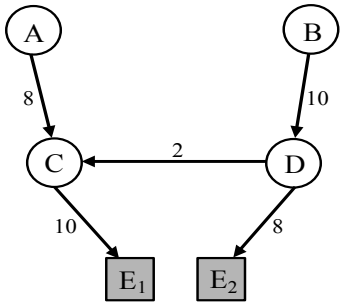


Figure 2.16: Optimal data flows per hour

Instead of relaying data through sensor D, sensor C now transmits all data from sensor A directly to the new sink E_1. Sensor D now relays a portion of data from B through C to E_1. The updated total energy usages per hour at sensors C and D are $T_C = T_D = 186$, while $T_A = 80$ and $T_B = 100$. The maximum WSN lifetime becomes $L_C = L_D = 500/186 \approx 2.7$ hours. An additional sink thus extends network lifetime from 2.3 to 2.7 hours.

Xue et al. [2005] extend their single-sink algorithm to determine optimal data flows in a WSN having multiple sinks. The algorithm first finds a shortest path from each sensor to a sink and determines the amount of flow on this path, based on data origination rates. The authors iteratively combine these paths to create a set of trees, each rooted at a sink. The running time of this algorithm thus remains the same as with the single-sink case, and the multi-sink adaptation of their algorithm achieves a $(1 - \epsilon)$-approximation in polynomial time.

Castao et al. [2013] propose a hybrid CG scheme to determine a schedule of data flows that

maximizes lifetime for multi-sink WSNs. In this problem, there exist a set of targets that are covered by a subset of sensors. Each column in the CG formulation corresponds to a subset of sensors that collectively cover the entire set of targets. As such, the column can be viewed as an in-flow tree rooted at a supersink node (to which all sinks can send flow), and each target location corresponds to a leaf node. To solve the CG subproblem, the authors propose a greedy randomized algorithm that uses a variation of depth-first search. If this algorithm fails to generate an attractive set of data flows, the authors solve an IP to generate such a set, or show that no positive reduced-cost column exists. The authors conclude that mixing their heuristic algorithm with the IP reduces computational time compared to a standard CG-based approach, and computes an optimal solution within the given time limit for most instances.

Whereas the previous works assume sink locations are established *a priori*, Kim et al. [2005] address how to optimally position multiple sinks from among the set of discrete sensor locations, in addition to finding an optimal set of data flows. The authors formulate mathematical models to address these two problems. They first consider an LP to compute optimal data flows from all sensors to a sink, in which sink locations are predetermined. The authors then formulate a mixed-integer program (MIP) to find an optimal set of sink locations and data flows, where an upper bound exists on the number of sinks. Similarly, Türkoğullari et al. [2010] formulate an MIP model to combine sensor and sink deployment and data-flow decisions. They provide a heuristic technique that first deploys sinks so that a sufficiently large number of sensors are within a minimum distance to a sink. Using these sink locations, the authors reformulate and solve the LP relaxation of the MIP using a CG scheme, where each column corresponds to a set of deployed sensors and data flows. Finally, they obtain a feasible solution to the MIP using an optimal set of columns from the relaxation.

2.4.2 Mobile-Sink WSNs

As an alternative to deploying multiple stationary sinks, sink mobilization extends network lifetime by balancing energy consumption among a larger set of sensors. In such cases the sink is permitted to move among a set of stationary locations at a predetermined speed. Mobile sinks are also useful in dynamic sensing environments that change over time as new targets emerge in the network [Khan et al., 2014].

2.4.2.1 Mobile-sink models with no transmission delay

Papadimitriou and Georgiadis [2006] introduce the following LP to address the *mobile-sink lifetime maximization problem* (MSM) in which $\mathcal{L}$ is the set of possible sink locations, and where the time required for the sink to move from one location to another is assumed to be negligible. The variables and parameters for this problem are the same as in model (2.1), except that the dwelling times per sink location and data flows are now indexed by the sink location $\ell \in \mathcal{L}$.

$$\max \sum_{\ell \in \mathcal{L}} z_\ell \tag{2.4a}$$

$$\text{s.t.} \sum_{j \in \mathcal{N}} y_{ij}^\ell - \sum_{h \in \mathcal{S}} y_{hi}^\ell - b_i z_\ell = 0 \quad \forall i \in \mathcal{S},\ \ell \in \mathcal{L} \tag{2.4b}$$

$$\sum_{\ell \in \mathcal{L}} \left[\sum_{j \in \mathcal{N}} e_{ij} y_{ij}^\ell + \sum_{h \in \mathcal{S}} c_{hi} y_{hi}^\ell \right] \leq P_i \quad \forall i \in \mathcal{S} \tag{2.4c}$$

$$y_{ij}^\ell \geq 0 \quad \forall i \in \mathcal{S},\ j \in \mathcal{N},\ \ell \in \mathcal{L} \tag{2.4d}$$

The objective function (2.4a) maximizes WSN lifetime defined as the sum of the sink dwelling times at all possible locations, and all other constraints are analogous to those in (2.1). An optimal solution to (2.4) provides the set of sink dwelling times and data flows to those sink locations.

While (2.4) determines both sink dwelling times and data flows, Wang et al. [2005] formulate an LP solely to determine sink movements and dwelling times in which sinks must be located at a corner point of a cell within a two-dimensional grid. Instead of determining data flow between nodes as in (2.4), they implement a shortest-path algorithm for data originating at each sensor. The authors observe that the network lifetime of mobile-sink WSNs increases by almost five times compared to a WSN having a single static sink.

Rather than assuming that the sink moves along a set predetermined locations, Huang et al. [2015] propose a reactive sink relocation method for hexagonal-grid WSNs. When the average residual energy within each cell reaches a minimum threshold, the sink moves to a neighboring cell having the highest average residual energy. Once the sink relocates to a new position, their method employs a shortest-path algorithm to determine data flows, in which edge weights are based on residual energy levels and data transmission size. Results show that relocating sinks according to a hexagonal grid-like topology can double the network lifetime in most cases compared to a static-sink WSN.

Whereas Papadimitriou and Georgiadis [2006] implements a centralized solution technique

by solving model (2.4), Gatzianas and Georgiadis [2008] assume decentralized coordination by using dual decomposition. The authors introduce locally-stored Lagrangian multipliers for energy-capacity and peak transmission constraints to formulate the Lagrangian dual problem. Taking advantage of the dual's special structure, they split this problem into a set of smaller subproblems and use a minimum-cost flow algorithm to solve the subproblem at each sensor to obtain a set of data flows. The authors show that their algorithm runs in polynomial time and results in maximum lifetime values within 3% of solutions using centralized approaches [Papadimitriou and Georgiadis, 2006].

In some cases, sink movement forces sensors to use excessively many hops to transmit data to a sink. To mitigate this problem, Liang and Luo [2011] jointly determine the sink trajectory and dwelling times with upper bounds limiting the distance traveled by each sink and the number of allowable hops for data transmission. They prove this problem to be NP-hard and consequently present a heuristic to identify a near-optimal solution. The algorithm first calculates the maximum potential dwelling time at each sink location. Next, the algorithm searches for a high-quality sink trajectory and determines the actual dwelling times at each location based on this trajectory.

2.4.2.2 Delay-tolerant models

Algorithms in the previous subsection assume that the time required for the sink to move from one location to another is negligible, and that sensors cannot delay transmission of their flows to a sink. In some cases, though, sensors may also be able to locally store data to delay transmission until the sink arrives at a more favorable location. We discuss these delay-tolerant models in this subsection, along with situations in which the travel time between sinks is non-negligible.

When data transmission can be delayed indefinitely, single mobile-sink and multiple stationary-sink WSNs yield identical optimal solutions, regardless of the sink's travel time. Similarly, this equivalence also holds when the sink can move infinitely fast and data can be delayed for any positive period of time (assuming negligible data transmission speeds). In more general cases, such as when data transmission cannot be delayed indefinitely, multiple stationary-sink WSNs yield lifetimes that are no less than those given by single mobile-sink WSNs.

For sinks having a fixed finite travel speed, maximum WSN lifetime is a nondecreasing function of maximum tolerable delay D, and reaches the maximum lifetime for its multiple stationary-sink WSN counterpart when D becomes sufficiently large. When the travel time between two sink locations is greater than D, the sink cannot move between these locations. The reason for this

restriction is that while the sink traverses from one location to the other, data is being collected and stored for delivery to the latter sink location. If the distance between these locations is too large, then the sink cannot arrive to the destination location within the delay limit. Thus, when D is less than the travel time between any two sink locations, the sink must remain stationary, and the problem reduces to choosing the best solution among several single-sink problems, one for each possible sink location.

Yun and Xia [2010] consider single mobile-sink WSNs with a finite positive maximum tolerable delay but negligible sink travel time. The authors extend (2.4) to propose a pair of LP models for the *delay-tolerant mobile sink lifetime maximization* (DT-MSM) problem, each with a unique variation of delay restriction. In the first variation, the sub-flow-based model, sensors are only allowed to store data gathered locally. For the second variation, the queue-based model, sensors may store data gathered by any sensor. The sub-flow-based formulation is similar to a standard multicommodity flow problem, which is solvable by fast, specialized algorithms [Assad, 1978, Kennington, 1978]. The queue-based model is more difficult to solve, but its increased flexibility leads to longer lifetimes than the sub-flow-based model.

Similar to the approach in [Gatzianas and Georgiadis, 2008], Yun et al. [2010a] implement a distributed algorithm for the DT-MSM problem, in which they examine the dual decomposition of the LP in [Yun and Xia, 2010] with Lagrangian multipliers on the flow-balance constraints. The authors split this problem into smaller problems at each sensor using information local to the sensor and its neighbors. They employ a fractional knapsack algorithm to separately solve each subproblem for the sub-flow-based and queue-based models. Based on analyzing a Lyapunov drift, the authors prove that this algorithm converges to an optimal solution.

Behdani et al. [2012] also take an exact solution approach to the queue-based DT-MSM problem in [Yun and Xia, 2010] and develop a partially-distributed CG algorithm. To identify an attractive set of data flows, the authors exploit the uncapacitated nature of WSN arcs and form separable shortest-path problems with non-negative costs at each sensor. While more efficient, this approach is only partially distributed, because the CG scheme requires the communication and solution of a master problem.

Whereas the previous approaches assume the sink travel time between locations to be negligible, Keskin et al. [2011] propose two MIP models to determine a sink tour that maximizes the lifetime of a WSN having a non-negligible sink travel time. The authors allow the sink to complete

multiple identical tours and place an upper bound on the duration of the sink tour to enforce a maximum tolerable delay. For the first model, gathered data is transmitted to the sink with no restrictions on the number of transmission hops, while the second model places an upper bound of the number of hops. To determine data flows in both cases, the authors implement a shortest-path algorithm based on residual energy levels. Rather than solving the MIP, the authors propose a set of heuristics to find a good feasible sink tour.

Alternatively, Behdani et al. [2013] determine a sink traversal tour and a set of data flows to maximize the lifetime of a WSN with a finite maximum tolerable delay and non-negligible sink travel time. For this problem, the authors allow sensors to transmit data to some location ℓ even when the sink is not currently dwelling at, or traveling to, ℓ. The sink must thus arrive at ℓ no more than D time units after a sensor begins transmission to ℓ. The goal is to find a sink tour over a subset of $\mathcal{L}$, and determine data flows to each location visited by the sink.

Because their model is important to the discussion in this section, we cover it in detail below. The authors first define $t_{\ell m}$ to be the time it takes the sink to travel from $\ell \in \mathcal{L}$ to $m \in \mathcal{L}$. Thus, the sink can only travel on arcs that satisfy $t_{\ell m} \leq D$, resulting in a directed sink graph $\mathcal{G}' = (\mathcal{L}, \mathcal{A})$ with an arc set $\mathcal{A} = \{(\ell, m) : t_{\ell m} \leq D\}$. For convenience, define node sets $K_\ell^+ = \{m \in \mathcal{L} : (\ell, m) \in \mathcal{A}\}$ and $K_\ell^- = \{m \in \mathcal{L} : (m, \ell) \in \mathcal{A}\}$.

Let $\bar{P}_i = P_i/C$ be the energy capacity of each sensor i during a sink tour, where C corresponds to the number of tours the sink must complete. Variable r_ℓ represents the period of time during which sensors transmit data to location ℓ. Let v_ℓ and $u_{\ell m}$ be binary variables that equal 1 if location ℓ is visited by the sink and if the sink travels along arc $(\ell, m) \in \mathcal{A}$, respectively. Therefore, the sink's dwelling time at ℓ must be at least $t_{m\ell}$ if $u_{m\ell} = 1$. Let the binary variable s_ℓ equal 1 if the sink tour originates at ℓ. Also, define $q_{\ell m} = s_\ell \times$ [the amount of time that sensors transmit data to m before the sink begins movement toward m from its preceding sink location]. Thus, if $s_\ell = 1$, then data is transmitted to m for $q_{\ell m} + \sum_{k \in \mathcal{K}_m^-} t_{km} u_{km}$ time units before the sink arrives at m. Letting M be a large constant value, the DT-MSM problem with a finite tolerable delay and non-negligible travel time can be formulated by the following NLP model [Behdani et al., 2013].

$$\max \sum_{\ell \in \mathcal{L}} r_\ell \tag{2.5a}$$

$$\text{s.t. } r_\ell \leq M v_\ell \quad \forall \ell \in \mathcal{L} \tag{2.5b}$$

$$\sum_{\ell\in\mathcal{L}}\left(\sum_{j\in\mathcal{N}} e_{ij}^{\ell}+\sum_{j\in\mathcal{S}} c_{ji}y_{ji}^{\ell}\right)\leq \bar{P}_i \quad \forall i\in\mathcal{S} \tag{2.5c}$$

$$\sum_{j\in\mathcal{N}} y_{ij}^{\ell}-\sum_{j\in\mathcal{S}} y_{ji}^{\ell}-b_i r_\ell=0 \quad \forall i\in\mathcal{N},\ \ell\in\mathcal{L} \tag{2.5d}$$

$$\sum_{m\in\mathcal{K}_\ell^+} u_{\ell m}=v_\ell \quad \forall \ell\in\mathcal{L} \tag{2.5e}$$

$$\sum_{m\in\mathcal{K}_\ell^-} u_{m\ell}=\sum_{m\in\mathcal{K}_\ell^+} u_{\ell m} \quad \forall \ell\in\mathcal{L} \tag{2.5f}$$

$$\sum_{\ell\in\mathcal{T}}\sum_{m\in\mathcal{T}} u_{\ell m}\geq v_k+v_r-1 \quad \forall\mathcal{T}\subset\mathcal{L}:2\leq|\mathcal{T}|\leq|\mathcal{L}|-2,\ k\in\mathcal{T},\ r\in\mathcal{T} \tag{2.5g}$$

$$\sum_{\ell\in\mathcal{L}} s_\ell=1 \tag{2.5h}$$

$$s_\ell\leq v_\ell \quad \forall \ell\in\mathcal{L} \tag{2.5i}$$

$$q_{\ell m}+\sum_{k\in\mathcal{K}_m^-} t_{km}u_{km}\leq D \quad \forall \ell\in\mathcal{L},\ m\in\mathcal{L} \tag{2.5j}$$

$$q_{\ell m}\geq\sum_{k\in\mathcal{K}_m^-}\left(q_{\ell k}+\sum_{h\in\mathcal{K}_k^-}(t_{hk}-r_k)u_{hk}\right)u_{km}-D(1-s_\ell) \quad \forall \ell\in\mathcal{L},\ m\in\mathcal{L} \tag{2.5k}$$

$$D(1-s_\ell)+r_\ell\geq q_{\ell\ell}+\sum_{m\in\mathcal{K}_m^-} t_{m\ell}u_{m\ell} \quad \forall \ell\in\mathcal{L} \tag{2.5l}$$

$$v_\ell\in\{0,1\} \quad \forall \ell\in\mathcal{L} \tag{2.5m}$$

$$u_{\ell m}\in\{0,1\} \quad \forall \ell\in\mathcal{L},\ m\in\mathcal{K}_\ell^+ \tag{2.5n}$$

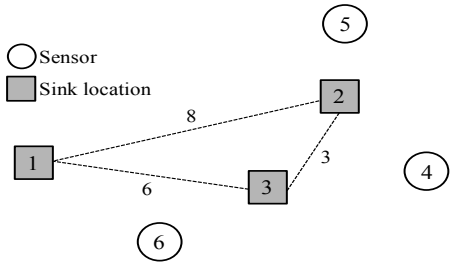
$$q_{\ell m}\geq 0 \quad \forall \ell,m\in\mathcal{L} \tag{2.5o}$$

$$r_\ell\geq 0 \quad \forall \ell\in\mathcal{L} \tag{2.5p}$$

The objective function (2.5a) maximizes the total time data is transmitted to any sink location $\ell\in\mathcal{L}$ while constraints (2.5b) ensure that data can only be transmitted to location ℓ if visited by the sink. Constraints (2.5c) and (2.5d) enforce energy-capacity and flow-balance constraints, respectively. Constraints (2.5e)–(2.5g) ensure that the sink's traversal corresponds to a single tour that visits all nodes ℓ such that $v_\ell=1$. Note that the exponential set of constraints (2.5g) in particular serve as subtour elimination constraints. Constraints (2.5h) and (2.5i) guarantee that one node is selected as the origin, from among those nodes ℓ for which $v_\ell=1$. Constraints (2.5j) and (2.5k) ensure that data transmitted to location $m\in\mathcal{L}$ is not delayed longer than D time units if $v_m=1$, while (2.5l)

enforces a similar maximum tolerable delay restriction at the origin location. Finally, constraints (2.5m) and (2.5n) enforce binary constraints on variables v and u, while constraints (2.5o) and (2.5p) enforce non-negativity of variables q and r. To solve this problem, the authors add a set of auxiliary variables to linearize model (2.5), and add the subtour elimination constraints (2.5g) as needed in a branch-and-cut fashion.

To illustrate the relationship between maximum lifetime and maximum tolerable delay, we consider a single mobile-sink delay-tolerant WSN lifetime maximization example. For this example the sink can move among locations 1, 2, and 3 traveling at a fixed speed of 1 km per hour. The arc labels in Figure 2.17a depict the distances between sink locations, and the table in Figure 2.17b displays the distances between sensors and sink locations. The energy to send one unit of data from a sensor to a sink location is equal to the distance between the two nodes. Each sensor possesses a battery power of 500 units, and the data generation rate for each sensor is 1 data unit per hour.



(a) Distance between sink locations

	Sink location		
Sensor	1	2	3
4	13	6	7
5	10	4	14
6	4	12	3

(b) Distance between sensors and sink locations

Figure 2.17: Distances (in km) for the single mobile-sink WSN example

Figure 2.18 shows how maximum WSN lifetime grows as a function of maximum tolerable delay. In general, WSN lifetime is nondecreasing in terms of delay, but this function might be nonconcave and even discontinuous, as is the case even in this simple example. When data cannot be delayed by three or more hours, the sink must remain stationary. Therefore, the sink remains at location 2 in Figure 2.17a for a maximum lifetime of 41.7 hours, which is longer than the maximum lifetime achieved if the sensor were stationary at locations 1 or 3. We thus vary the maximum tolerable delay parameter and solve (2.5) to determine the maximum WSN lifetime. Next, suppose that data can be delayed by three hours. The sink can then travel between locations 2 and 3 without violating the delay constraint. The maximum lifetime increases by 46% to 60.9 hours. As the maximum tolerable delay increases from three to six hours, the maximum lifetime linearly increases

as sensor 6 is able to transmit more data to the more favorable sink location 3, thus prolonging network lifetime.

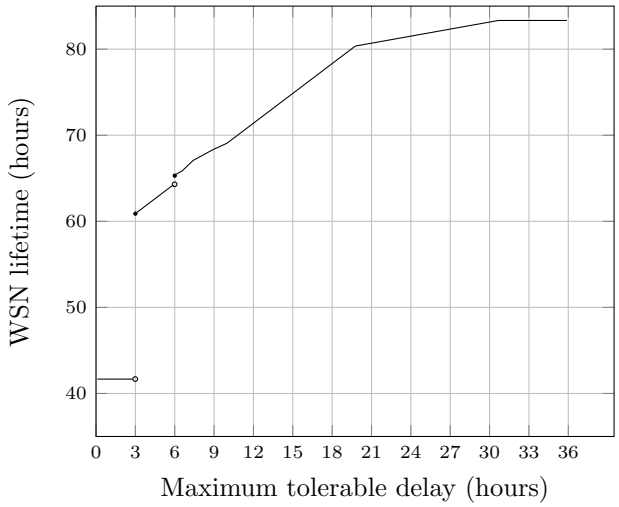


Figure 2.18: Maximum WSN lifetime as a function of maximum tolerable delay

The WSN lifetime also has a discontinuous jump when data can be delayed by 6 hours because the sink can then move between location 1 and location 3. As the maximum tolerable delay increases beyond 6 hours, each sensor is able to transmit more data to its nearest sink location to balance energy consumption to prolong network lifetime. When the maximum tolerable delay surpasses roughly 30.7 hours, the single mobile-sink WSN reaches the maximum lifetime for its multiple stationary-sink WSN counterpart and yields equivalent flows to the case when a sink is placed at each possible location.

2.4.3 Multiple Mobile-Sink WSNs

Just as independently adding and mobilizing sinks prolongs WSN lifetime, introducing multiple mobile sinks further balances energy consumption and prolongs WSN lifetime. Gandham et al. [2003] present a sink-relocation and data-routing approach for multiple mobile-sink WSNs. The authors first assume that a limited number of sinks dwell at some set of locations for a predetermined period of time until they are relocated to a new set of locations. Assuming that the sink relocation time is negligible, they formulate an IP model to determine this new set of locations from among a set of predefined discrete locations, along with data flows from the sensors to one of the sink locations, in a manner that minimizes the maximum energy consumption among all sensors. They

Author (Model)	Problem specification	Technique
	Multiple mobile-sink models	
Gandham et al. [2003]	Periodic sink relocation and routing	Rounding heuristic to solve an IP that minimizes the maximum energy consumption among all sensors
Basagni et al. [2008]	Sink relocation and routing	Decompose LP model to add violated dual constraints by solving a p-median problem
Basagni et al. [2009]	Distributed sink relocation and routing	Allow sensors to communicate with neighboring sensors to improve network awareness
Marta and Cardei [2009]	Distributed reactive sink movement on a hexagonal grid	Sink relocates to an area having greater energy when neighboring sensors reach a low-energy threshold
Saad and Tourancheau [2009]	Multiple mobile sink control within buildings	IP with a minimum sink dwelling time, shortest-path algorithm for routing

Table 2.7: Section 2.4.3 summary, where sink travel times are negligible and routing cannot be delayed

heuristically solve the IPs by rounding fractional LP relaxation values, repeating the process until a sensor exhausts its energy.

Basagni et al. [2008] present a set of heuristics for determining the movement of multiple sinks in addition to data-routing decisions. The authors formulate and decompose an LP model to the multiple MSM problem to add violated dual constraints using a p-median problem [Mladenović et al., 2007]. Additionally, Basagni et al. [2009] formulate a distributed version of their heuristic that allows sensors to communicate with neighboring sensors to improve network awareness. Performance comparisons show that the distributed heuristic achieves near-optimal network lifetimes that provide significant improvements compared to random sink mobility and multiple static sink deployment.

Marta and Cardei [2009] take a reactive approach for sink mobility to create a heuristic where sinks periodically move along the perimeter of a hexagonal grid to form a virtual backbone. Each sink monitors its neighboring sensors. When a predefined percentage of these sensors reaches a low energy threshold, the sink moves to avoid burdening those sensors with relaying a large amount of data to the sink. The sink thus searches for nearby areas along the perimeter whose local sensors

have greater energy. Since sinks only utilize information within its neighborhood, this heuristic is implemented in a distributed manner. The authors also consider the case in which the sink moves along a non-predetermined path from one location to another.

Finally, as an application of these studies, Saad and Tourancheau [2009] examine the deployment of multiple mobile sinks in WSNs within buildings. The authors explore a clustering mechanism to decompose the WSN, using an IP to determine sink trajectories and dwelling times. Because frequently relocating sinks can be expensive, the authors require sinks to dwell at each location for a minimum time period. For local and global routing decisions, the authors implement a shortest-path algorithm that considers the distance between nodes and the residual energy capacities at each sensor. The authors compare three scenarios for their application: stationary sinks, multiple sinks moving within different clusters, and multiple sinks moving throughout the entire network.

2.5 Alternative Optimization Metrics

Section 2.5.1 explores techniques for maximizing the lifetime of the remaining network after a sensor exhausts its energy. In Section 2.5.2 we examine algorithms for maximizing the coverage area and connectivity of a WSN. We explore approaches for minimizing data transmission delays in Section 2.5.3. Table 2.8 summarizes the papers discussed in this Sections 2.5.1 and 2.5.2, and Table 2.9 summarizes those discussed in Section 2.5.3.

2.5.1 Maximizing Conditional Sensor Lifetime

For this discussion, given a feasible solution to problem (2.1) represented by variables $\hat{\mathbf{y}}$ and $\hat{z}$, the lifetime of sensor $i \in \mathcal{S}$ is given by

$$\frac{P_i \hat{z}}{\left(\sum_{j \in \mathcal{N}} e_{ij} \hat{y}_{ij} + \sum_{h \in \mathcal{S}} c_{hi} \hat{y}_{hi}\right)}. \tag{2.6}$$

Suppose that for some feasible $\mathbf{y}$, we sort these sensor lifetimes in nondecreasing order, yielding lifetimes $L_1(\mathbf{y}), \ldots, L_{|\mathcal{S}|}(\mathbf{y})$ (where we omit $\hat{z}$ from this notation for simplicity). Formulation (2.1) maximizes $L_1(\mathbf{y})$, with no regard to $L_j(\mathbf{y})$, for $j \geq 2$. Typically, though, multiple optimal solutions exist to (2.1). Defining $L_0(\mathbf{y}) = 0$ for any solution $\mathbf{y}$, the jth conditional lifetime is the maximum possible value of $L_j(\mathbf{y})$, over all solutions $\mathbf{y}$ feasible to the constraints in (2.1), such that $L_i(\mathbf{y})$ equals the ith conditional lifetime for all $i = 0, \ldots, j-1$. By optimizing the $|\mathcal{S}|$th conditional lifetime, we

Author (Model)	Problem specification	Technique
	Maximizing conditional sensor lifetime	
Dagher et al. [2007]	Conditional sensor lifetime	Iterative LP algorithm to identify a PO solution
Mansouri et al. [2008]	Conditional commodity lifetime	Solve $\|\mathcal{L}\|$ MIPs to determine PO solution
	Maximizing connectivity and coverage	
Yun et al. [2010b]	Connectivity-constrained sensor deployment	Provides relation to general set-covering problem with polygon-based methodology to prove optimality
Wang and Wu [2014]	Single-objective sensor deployment to avoid network fragmentation	Improves connectivity and coverage by deploying sensors at critical nodes and low-coverage areas
Pradhan and Panda [2012]	Sensor deployment for coverage and connectivity	Particle swarm optimization algorithm
Sengupta et al. [2013]	Sensor deployment/routing for coverage, lifetime, and energy use	GA to solve a series of weighted single-objective problems to find a PO solution
Sengupta et al. [2012]	Same as [Sengupta et al., 2013], with varying connectivity requirements	Decompose IP into subproblems and use GA to simultaneously solve the subproblems
Özdemir et al. [2013] (MOEA/D)	Cluster formation to balance energy consumption and coverage	Evolutionary algorithm that selects active sensors as clusterheads
Rossi et al. [2012b]	Max lifetime and min coverage breach under bandwidth limits	CG approach with a GA to solve the MIP subproblems
Rossi et al. [2012a]	Coverage/energy tradeoff when sensing ranges are adjustable	Formulations for two problems; CG approach with GAs to solve the MIP subproblems
Gentili and Raiconi [2013]	Single-objective data-routing with minimum coverage constraints	Greedy approach to initialize CG scheme, additional subproblem constraints to avoid redundant columns

Table 2.8: Section 2.5.1 and 2.5.2 summary

maximize $L_1(\mathbf{y})$, breaking ties by maximizing $L_2(\mathbf{y})$, and so on, with maximizing $L_{|\mathcal{S}|}(\mathbf{y})$ as the final criterion. We refer to this problem as the *conditional lifetime problem*.

Dagher et al. [2007] present an algorithm for solving the conditional lifetime problem. The algorithm tracks "fixed" sensors, which are sensors whose lifetime values have been determined. Let

E (initially empty) denote the set of all fixed sensors. Their algorithm iteratively solves a series of LPs, each of which contains the constraints in (2.1) plus constraints restricting the fixed sensors to take on their designated lifetime values. At each iteration, the algorithm enforces the condition that all lifetimes for sensors not in E must equal some common value ζ, and solves an LP to maximize ζ. Next, for every sensor j not in E, the algorithm solves individual LPs to maximize the lifetime of sensor j, subject to the constraints on the previous model, and constraints stating that every other sensor in E has a lifetime of at least ζ. If the maximum lifetime of each individual LP is ζ, for all $j \notin E$, then the current solution solves the conditional lifetime problem. If not, then the algorithm solves another LP for each sensor not in E to identify a sensor $\hat{\jmath} \notin E$ whose lifetime cannot exceed ζ. Sensor $\hat{\jmath}$'s lifetime is then fixed to ζ, and sensor $\hat{\jmath}$ is placed in E.

Mansouri et al. [2008] take a multi-commodity approach to maximize conditional lifetime, in which a commodity consists of all data originating at various targets destined for a common sink. A commodity's lifetime is given by the minimum lifetime among all sensors transmitting that commodity. Using a similar conditional lifetime definition as [Dagher et al., 2007], the authors propose an iterative algorithm that solves an MIP to maximize the first conditional commodity lifetime, in which binary variables specify whether a sensor transmits a specific commodity. Step $n \geq 2$ of the algorithm chooses a solution among those optimal in step $n-1$ that maximizes the next conditional commodity lifetime. The algorithm iteratively runs until the $|\mathcal{L}|$th conditional commodity lifetime has been maximized, where $|\mathcal{L}|$ is the total number of sinks.

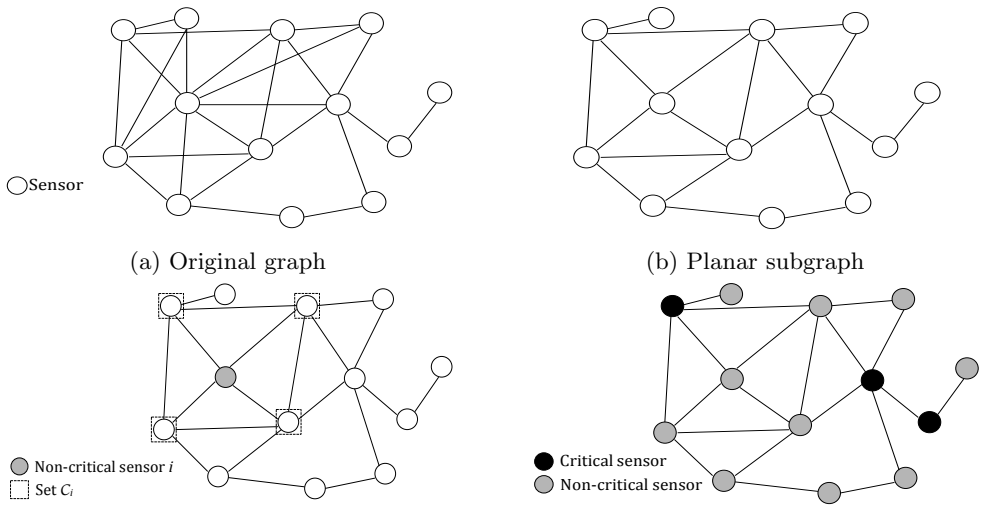
2.5.2 Maximizing Coverage and Connectivity

Sensor locations are not only vital in maximizing WSN lifetime, but are also critical in assuring that all targets are covered as desired. Optimizing lifetime and maximizing coverage are two different objectives, but can be considered in joint optimization studies as detailed in this section. Yun et al. [2010b] explore a proactive method to find the smallest set of deployed sensors that collectively monitors all target locations, under the condition that each sensor must be connected to k other nodes for some $k \leq 6$. They show how the sensor deployment problem relates to the general set-covering problem and use a polygon-based methodology to prove optimality.

Because the failure of a single sensor can potentially disconnect networks, Wang and Wu [2014] seek to improve WSN connectivity by deploying additional sensors within an existing network. In particular, they place sensors at critical nodes whose removal would partition the network into

two or more separate components.

Assuming that sensors correspond to a set of vertices located in Euclidean space (Figure 2.19a), the authors first create a planar subgraph of the WSN by removing a subset of crossing edges (Figure 2.19b). Next, for each sensor i, the algorithm finds its neighbors $\mathcal{C}_i$ in this planar graph. If there exist $|\mathcal{C}_i|$ 2-simplices involving i and a pair of nodes in $\mathcal{C}_i$, then i is deemed non-critical (see Figure 2.19c). (A 2-simplex is a two-dimensional polytope formed by the convex hull of three sensors, i.e., a triangle.) The algorithm then iteratively removes each of the remaining sensors to determine if it is critical. If so, a backup sensor is placed near the critical sensor (Figure 2.19d). Additionally, the authors implement a protocol to reduce the size of large 2-simplices in the subgraph that correspond to areas of low coverage in the WSN (Figure 2.20a). Their algorithm uses a localized method to identify these simplices and introduces a new sensor to reduce its size by eliminating at least one sensor its boundary (Figure 2.20b).

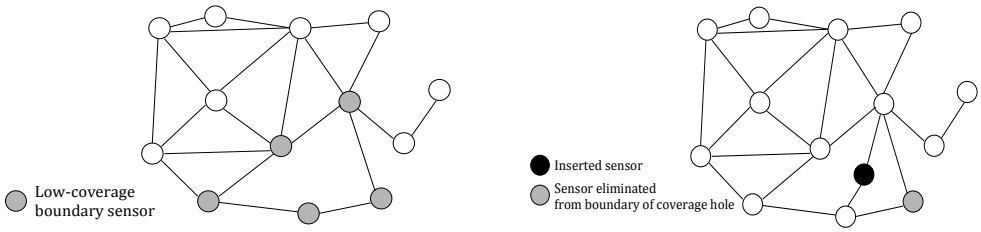


(a) Original graph (b) Planar subgraph

(c) Non-critical sensor (d) Critical and non-critical sensor graph

Figure 2.19: Locating critical sensors

While the prior literature in this section focuses on deploying sensors to maximize connectivity, Pradhan and Panda [2012] search for a Pareto-optimal (PO) solution that jointly considers coverage and lifetime. (A feasible solution x to a multi-objective problem is said to be PO if no other feasible solution exists that is better than x with respect to one objective and at least as good as x with respect to the other objectives [Censor, 1977].) The authors first define WSN coverage to be the percentage of target locations being monitored by at least one sensor. They then propose a multi-objective



(a) WSN coverage hole (b) WSN with coverage hole eliminated

Figure 2.20: Locating and reducing the size of a coverage hole

PSO algorithm in which each particle corresponds to a set of deployed sensors and data flows. After their algorithm initializes particle positions and fitness functions, each particle randomly moves through the solution space according to the fitness value of the best current solution. During each iteration, their algorithm compares newly-generated solutions, stores any non-dominated solutions, and updates associated fitness functions. The authors show that their algorithm produces a diverse set of solutions along the Pareto frontier of the two objectives.

Similarly, Sengupta et al. [2013] propose a sensor deployment and data-routing approach that (a) maximizes coverage, (b) maximizes network lifetime, and (c) minimizes energy consumption. The authors decompose this problem into a series of weighted single-objective optimization problems to find PO solutions. Sengupta et al. [2012] extend the work in [Sengupta et al., 2013] to maximize network lifetime and coverage area, in which different portions of the network require varying levels of connectivity. They decompose a multi-objective IP into a series of smaller weighted subproblems that are simultaneously solved using a GA.

Özdemir et al. [2013] use a weighted-sum approach to implement a decomposition-based multi-objective evolutionary algorithm (MOEA/D) to determine a set of sensor clusters and clusterheads that maximizes coverage and minimizes energy consumption. A solution using their MOEA/D algorithm contains a set of clusterheads and a set of active sensors that relay data to a clusterhead. In each iteration of the algorithm, clusterheads are selected from among the set of active sensors from the previous solution. Results show that the MOEA/D improves upon solutions provided by standard GAs when solving the clusterhead selection problem with a coverage maximization objective.

Another portion of literature focuses on optimizing a single objective while ensuring that secondary objectives meet some desired level. Rossi et al. [2012b] present a CG-based solution technique for two alternative WSN network problems that also optimize network coverage. The first

problem, minimizing coverage breach under bandwidth constraints (MCBB), determines a set of data flows that minimizes the number of uncovered targets given lifetime and bandwidth constraints. The second problem searches for a solution that maximizes network lifetime under coverage and bandwidth constraints (MNLB). The authors formulate an MIP for both problems, in which binary variables denote whether a sensor monitors some set of targets. The authors implement a CG algorithm to solve these models and present a GA to solve the MIP subproblem. If the GA reaches a local optimum, then the authors solve the MIP to find an attractive column or prove optimality of the current solution. The authors also consider a bi-objective problem to explore the trade-off between maximizing network lifetime and minimizing the number of uncovered targets. They solve a series of MCBB instances with varying levels of lifetime constraints to search for the Pareto frontier of the two objectives. Results show that mixing the GA and MIP to solve the MCBB and MNLB subproblems produces competitive results compared to strictly solving the MIP.

Rossi et al. [2012a] extend their work in [Rossi et al., 2012b] to maximize WSN coverage when sensors are capable of adjusting their coverage ranges to conserve energy. The authors present two MIP variations in which sensing ranges are continuously adjustable or are chosen from among a set of predetermined values. Similar to [Rossi et al., 2012b], they present a CG algorithm using a GA to solve the subproblem. In this case, results show that the GA decreases computational time over solving the subproblem exactly by branch-and-bound, without compromising solution quality.

Gentili and Raiconi [2013] also give a CG-based algorithm to determine a schedule of data flows that maximizes lifetime when some $(1 - \alpha)$ percentage of targets may be left uncovered. They implement a greedy approach to initialize their CG scheme and implement some regularity conditions to speed up convergence. Additionally, they add a set of constraints to the subproblem to avoid redundant columns. Compared to the case when all targets must be covered, the authors show that the lifetime increases by 48.2% for $\alpha = 85\%$.

2.5.3 Minimizing Transmission Delay

As described in Section 2.4.2.2, some WSNs permit delays in data transmission. While long delays may be permitted for some cases, other applications give higher priority to data transmitted to the sink in real-time. For such applications, researchers also consider minimizing total delay, defined as the time between when the data arrives at a sensor and when it arrives at a sink.

Author (Model)	Problem specification	Technique
	Minimizing Transmission Delay	
Shah-Mansouri and Wong [2007]	Distributed data-routing to max lifetime and min delay	Iterative LP algorithm, regularized method with dual decomposition to solve subproblems
Xu et al. [2012]	Sink traversal with delay limits	Set-covering formulation, in-tree for routing flows
Ammari [2013] (TED)	Data-routing to min and balance energy consumption and min delay	Decompose area around sensors to prioritize nearby sensors, weighted-sum model for routing
Shan et al. [2013]	Distributed backbone construction to min data transmission length	Balances energy consumption among sensors some number of hops away from the sink

Table 2.9: Section 2.5.3 summary

Shah-Mansouri and Wong [2007] examine the problem of finding a data-routing plan that maximizes network lifetime; furthermore, if multiple optimal solutions to this problem exist, then the procedure optimizes a secondary objective. The secondary objectives they consider are the minimization of transmission delay and the minimization of total energy consumption. One approach simply solves an initial LP to maximize network lifetime. Then, one can solve an additional LP to minimize the secondary objective, in which all sensors' lifetimes are constrained to be no less than the maximum lifetime value. As an alternative, the authors propose a regularization method to combine these two LPs into a single optimization problem. They present a regularization function corresponding to each secondary objective that can be added to the objective function of the initial LP to formulate a regularized optimization model. The authors show that as long as the coefficient on this regularization function is no greater than some positive threshold, a solution to this regularized model corresponds to an optimal solution using the iterative LP approach. Using dual decomposition, they decompose this model into subproblems located at each sensor that can be solved in a distributed manner. The authors show that this regularized approach maximizes network lifetime and provides shorter transmission delays and lower energy consumption compared to the algorithms in [Chang and Tassiulas, 1999] and [Madan and Lall, 2004].

Xu et al. [2012] examine the trade-off between network lifetime and transmission delay for the DT-MSM problem (see Section 2.4.2.2). The authors develop a heuristic that determines a mobile

sink trajectory and routing protocol with varying bounds on maximum delay. At each iteration of the algorithm, a new sink location is added to the trajectory as long as delay bounds are not violated. Sink locations are chosen with respect to the total number of sensors that neighbor the sink location, and the distance between each sensor and the sink location. The algorithm then builds an in-tree of data flows at each sink location so as to minimize total energy consumption. Additionally, the authors propose two heuristics to determine the sink trajectory based on a set-covering formulation.

Ammari [2013] propose a multi-objective optimization approach to determine a set of data flows that minimizes energy consumption, balances energy consumption among the sensors, and minimizes transmission delay. To solve this problem, the authors propose a data-forwarding protocol, trade-off with energy delay (TED), to find an appropriate trade-off among these three objectives. This protocol first decomposes the area around each sensor in such a way that gives priority to closer sensors when relaying data transmissions. To determine data flows, they formulate a weighted-sum model to optimize over the three objectives.

In some applications, data transmission must be delayed at a node until all data from neighboring nodes is received. Additionally, data originating at sensors located at nodes in a routing tree may require a large number hops for transmission. In these cases data transmission may accumulate significant delays due to paths that have a large number of hops. Shan et al. [2013] take a centralized approach to this problem. Their approach builds a backbone that maximizes network lifetime while minimizing the length of data-flow paths from each sensor to a sink. The algorithm heuristically constructs a routing tree that balances energy consumption among all sensors that are at least some number of hops away from the sink. They propose a distributed refinement algorithm that exchanges information between neighboring nodes to balance the energy consumption among the children of each node.

2.6 Future Challenges

This chapter reveals many open areas for future research. One key assumption that may not be satisfied in practice regards the *a priori* knowledge of instance data. Demands originating at targets (or sensors) would appear to be far less certain. The online approaches in Section 2.3.1 consider uncertain or unknown data demand distributions, while most other studies in the literature assume known uniform demands. We recommend research targeted towards situations in the middle,

where some imperfect knowledge of demands is available. In these cases, future research could employ stochastic programming [Higle and Sen, 1991, Mulvey and Ruszczyński, 1995, Neely, 2010] or robust optimization [Ben-Tal and Nemirovski, 2002, Mulvey et al., 1995, Romich et al., 2015] principles to determine data flows in order to achieve an expected maximum lifetime, or worst-case maximum lifetime, as appropriate.

The previous works assume that data transmissions are free of interference that inhibit successful transmission and lead to low-quality or unreliable data. For practical implementation, future research needs to consider the presence of stochastic interference levels. Multi-path or redundant data transmissions may be viable options to embed interference countermeasures into network protocols. Similarly, works in this chapter assume that network coordination and operation are free of attacks. Since WSNs often transmit data on public channels, they are susceptible to attacks on transmission links [Das, 2009]. Similarly, sensors are prone to physical attacks since they are commonly left unattended after deployment. Countermeasures for such attacks could be modeled as a network interdiction problem [Golden, 1978, Janjarassuk and Linderoth, 2008, Smith et al., 2013], in which a WSN operator maximizes lifetime after an adversary attacks some set of links. Additionally, this problem could be extended to a three-stage model in which the WSN operator fortifies the WSN (e.g., relocating sensors, alleviating bandwidth restrictions) before an adversary attacks.

While the approaches in this chapter seek to balance energy consumption among all sensors, WSN lifetime is still constrained by finite sensor battery power. Some WSNs allow for energy harvesting [Gilbert and Balouchi, 2008, Kansal et al., 2007, Visser and Vullers, 2013, Vullers et al., 2010], in which sensors gather energy by means of various mechanisms (e.g., solar power, radio frequency energy transfer, microwave energy transfer) to recharge their battery. Future optimization approaches need to address how to utilize these energy sources to prolong network lifetime. Employing such techniques also opens up the discussion for multiple optimization problems, such as when and how to harvest energy to sustainably operate a WSN.

Data flows are predominantly assumed to be non-negative continuous values, but for some applications, positive data-flow values less than some minimum threshold may be undesirable. To characterize such cases, future optimization approaches may utilize semicontinous variables that, if positive, must be no less than some lower bound. Another common assumption is that data is transmitted (virtually) instantaneously from sensors to other nodes in the network. If this transmission

time is non-negligible, then for problems having tolerable delays, the transmission times from the data origin to its destination must be taken into account. For mobile-sink WSN models, the sink may need to dwell at each location for some minimum amount of time to ensure that all data transmissions are retrieved.

Clustering techniques presented in Section 2.3.2 help to decentralize network protocols and coordination. A majority of the presented work propose heuristic techniques for the clusterhead selection and cluster formation problems. While Nikolidakis et al. [2013] and Kuila and Jana [2014] propose LP models for a special case of the clusterhead selection problem, future work could employ mathematical programming techniques for the general clusterhead selection and cluster formation problems. This work might examine exact decomposition-based solution techniques, such as column generation and Langrangian relaxation.

Typically, sinks are assumed to exist at specified discrete locations, although some research also regards the placement of sensors and sinks from among a set of discrete candidate locations. The placement of sensors in a more general Euclidean space is a very challenging research problem that warrants investigation. Additionally, if multiple sensor deployment is infeasible or too costly, sensor mobilization may be a viable solution. Future research could consider the presence of a mobile sensor that moves along a set of predetermined discrete locations. Similar to mobile-sink models, a mobile-sensor lifetime maximization model might consider determining a sensor's trajectory and dwelling times at each location. Furthermore, in mobile-sink examples, the speed of the sink is exogenous, and does not utilize any resources within the system. An alternative model regards the case in which sinks have energy too, and this energy is consumed by receiving data and by moving from one location to another. Moreover, the rate at which this energy is consumed would be a function of the sink's speed. Such a model could examine the alternative of maximizing the sink lifetime.

Finally, we submit that many future WSN studies will be guided within context-specific applications. The algorithms presented in this chapter are mostly targeted toward models that have broad applicability. When implementing these algorithms for specific applications, certain aspects of the network will need to be tailored for the situation at hand, revealing even richer classes of problems for study.

Chapter 3

Augmenting-flow Algorithms for Solving a Class of Maximum Flow Problems Having Node-capacity Restrictions

3.1 Introduction and Problem Statement

In this chapter we study a variation of the maximum flow problem (MFP) having a set of node-capacity restrictions. This problem takes place on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ with a node set $\mathcal{V}$ and an arc set $\mathcal{A}$. Set $\mathcal{V}$ contains a source node s and a sink node t. As usual, each arc $(i, j) \in \mathcal{A}$ has a capacity of $c_{ij} > 0$, which refers to the maximum amount of flow that can be sent on (i, j). In this chapter we examine the case in which each unit of flow on (i, j) consumes $g_{ij} > 0$ units of capacity at node $i \in \mathcal{V}$, and define $b_i > 0$ to be the capacity of node i.

To formulate this problem, let scalar variable z refer to the maximum flow through the network, and let non-negative variables x_{ij} correspond to the amount of flow on arc $(i, j) \in \mathcal{A}$. The node-capacitated maximum flow problem (NCMFP) seeks to send as much flow as possible from s to t without violating any node- or arc-capacity constraint. The NCMFP can be formulated by the following linear programming (LP) model.

$$\max z \tag{3.1a}$$

$$\text{s.t.} \sum_{i:(s,i)\in\mathcal{A}} x_{si} = z \tag{3.1b}$$

$$-\sum_{j:(j,t)\in\mathcal{A}} x_{jt} = -z \tag{3.1c}$$

$$\sum_{j:(i,j)\in\mathcal{A}} x_{ij} - \sum_{h:(h,i)\in\mathcal{A}} x_{hi} = 0 \quad \forall i \in \mathcal{V}\setminus\{s,t\} \tag{3.1d}$$

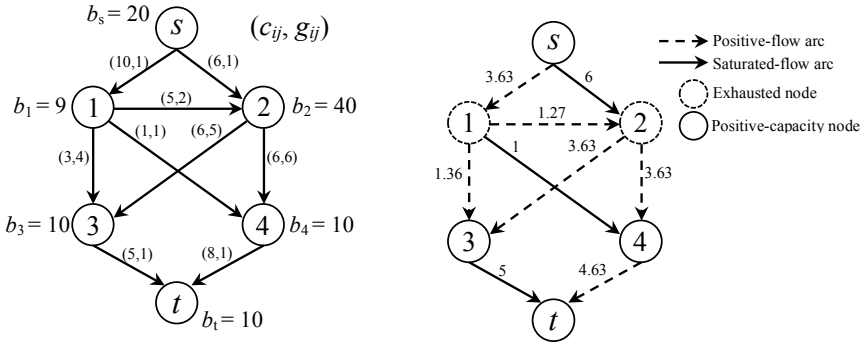
$$\sum_{j:(i,j)\in\mathcal{A}} g_{ij}x_{ij} \le b_i \quad \forall i \in \mathcal{V}\setminus\{t\} \tag{3.1e}$$

$$0 \le x_{ij} \le c_{ij} \quad \forall (i,j) \in \mathcal{A} \tag{3.1f}$$

The objective function (3.1a) and constraints (3.1b) and (3.1c) maximize flow from s to t, and constraints (3.1d) ensure flow balance among all nodes in $\mathcal{V}\setminus\{s,t\}$. Constraints (3.1e) and (3.1f) represent node- and arc-capacity constraints, respectively. Finally, constraints (3.1f) also enforce flow non-negativity.

For the special case in which there exist constants g_i, such that $g_{ij} = g_i,\ \forall (i,j) \in \mathcal{A}$, the NCMFP can be simplified to the MFP by first splitting each node $i \in \mathcal{V}$ into two nodes i_1 and i_2, and adding an arc $(i_1, i_2) \in \mathcal{A}$, with $c_{i_1 i_2} = b_i/g_i$. All arcs $(h,i) \in \mathcal{A}$ are revised to (h, i_1), and all arcs $(i_2, j) \in \mathcal{A}$ are revised to (i_2, j). Ford and Fulkerson [1958] utilize this transformation when $g_{ij} \in \{0,1\},\ \forall (i,j) \in \mathcal{A}$. On a similar note, the NCMFP can also accommodate the case in which flow on arc (i,j) consumes capacity at node j, if flows on each arc (i,j) consume an identical amount of capacity at node j. Suppose that for all $i \in \mathcal{V}:\ (i,j) \in \mathcal{A}$, flow on (i,j) consumes $h_j > 0$ units of capacity at node j. We can transform this model to the NCMFP by adjusting $g'_{jk} = g_{jk} + h_j,\ \forall (j,k) \in \mathcal{A}$, and bounding z by b_t/h_t if b_t is finite.

In this chapter we address the more general statement of the NCMFP having $g_{ij} > 0,\ \forall (i,j) \in \mathcal{A}$, in which the aforementioned transformations cannot be applied. Figure 3.1 illustrates an NCMFP instance. Figure 3.1a displays all data for this example, while Figure 3.1b displays an optimal set of flows that maximizes flow from s to t without violating any node- or arc-capacity restrictions. Each dashed arc (i,j) refers to a positive non-saturated flow $0 < x_{ij} < c_{ij}$ on (i,j), while each solid arc (i,j) refers to a saturated flow $x_{ij} = c_{ij}$ on (i,j). Not surprisingly, the maximum flow-minimum cut theorem [Ahuja et al., 1993] for the MFP does not hold for the NCMFP. Figure 3.1b demonstrates this claim since there exists a path $(s,1,2,4,t)$ of non-saturated arcs from s to t. (Clearly, however, if a cut set of saturated arcs does not exist, then every s–t path containing only non-saturated arcs in an optimal solution must visit at least one node whose capacity is exhausted.) Thus, traditional augmenting-flow algorithms (e.g., Ford-Fulkerson [Ford and Fulkerson, 1956] and push-relabel [Ahuja



(a) Network flow example having node and arc capacities.

(b) Optimal flows x

Figure 3.1: Node-capacitated network flow example, in which $b_s = b_3 = b_4 = b_t = \infty$.

et al., 1993]) generally will not optimize the NCMFP.

In the absence of constraints (3.1e), model (3.1) can be used to solve the MFP having only arc-capacity constraints. To avoid the computational difficulties associated with solving a linear program, many efficient augmenting-flow algorithms have been proposed to solve the MFP. While some algorithms augment flow on all arcs along s–t paths (e.g., Ford-Fulkerson [Ford and Fulkerson, 1956]) during each augmentation, other approaches augment flows on individual arcs (e.g., push-relabel). These algorithms do not allow flows to violate capacity constraints, and terminate when it can be verified (sometimes indirectly) that every s–t path includes a saturated arc in the current solution. One of the key observations in this chapter is that an analogous criterion — that every such path *either* visits a saturated arc *or* a node whose capacity is exhausted — is not sufficient to prove optimality. Generally speaking, there may exist a flow circulation that increases the available capacity on a set of exhausted nodes, thus permitting additional flow to be transmitted from s to t (see Section 3.2.3).

Depending on the network size and data characteristics, solving (3.1) could become time-consuming. Our primary contributions are thus two augmenting-flow algorithms that avoid solving (3.1) for the NCMFP. Both algorithms modify an augmenting-flow algorithm to obtain a feasible NCMFP solution. Rather than solving (3.1), the first algorithm implements two smaller auxiliary LPs to solve the NCMFP. These two LPs either prove the optimality of a given NCMFP solution, terminate with a feasible NCMFP solution, or augment flow first on a circulation to increase some node capacities and then on an s–t path to maximize additional flow in the network. This first algorithm tends to encounter small, easily solvable LPs; however, some situations might call for the

complete avoidance of linear programming subroutines. Accordingly, our second approach is heuristic only, and modifies the circulation-generation part of the first approach to obviate the necessity of solving an LP.

This rest of this chapter is organized as follows. In Section 3.2, we present our almost-exact augmenting-flow algorithm for the NCMFP. We next detail a heuristic version of our augmenting flow algorithm in Section 3.3. Finally, Section 3.4 presents computational results for both augmenting-flow algorithms.

3.2 Augmenting Path and Cycle Algorithm

In this section we describe our *almost-exact* Augmenting Path and Cycle (APC) algorithm for solving the NCMFP. We refer to the APC as almost-exact because, in most cases, it obtains a feasible NCMFP solution for which we can prove optimality. In all other cases, the APC produces flows close to optimal for the NCMFP. We first describe the APC initialization phase in Section 3.2.1 that determines an initial set of feasible flows $\bar{x}$. Next, Section 3.2.2 details a dual-based heuristic algorithm for checking the optimality of $\bar{x}$. When flows $\bar{x}$ are not deemed optimal, we implement the APC augmenting-flow phase in Section 3.2.3 to either prove the optimality of flows $\bar{x}$, terminate the APC with feasible flows $\bar{x}$, or to identify and adjust flow on a set of augmenting cycles and an s–t path. We also describe the situations in which the APC produces sub-optimal solutions. Finally, in Section 3.2.4 we prove the conditions in which the APC produces flows $\bar{x}$ optimal to the NCMFP.

3.2.1 APC Initialization Phase

To obtain an initial feasible solution to (3.1), we execute a modified version of the augmenting-flow (AF) algorithm [Ahuja et al., 1993] for solving the MFP. We call our modified AF algorithm the *m-AF algorithm*. Letting set $\mathcal{A}'$ contain all arcs in the residual network of $\mathcal{G}$ given flows $\bar{x}$ (i.e., $\mathcal{A}' = \{(i,j) : (i,j) \in \mathcal{A} \text{ and } \bar{x}_{ij} < c_{ij}, \text{ or } (j,i) \in \mathcal{A} \text{ and } \bar{x}_{ij} > 0\}$), the standard AF first determines an s–t path p containing only arcs in $\mathcal{A}'$. Alternatively, the m-AF searches for an s–t path $\bar{p}$ that contains only reverse arcs $(u,v) \in \mathcal{A}' \setminus \mathcal{A}$ or forward arcs $(i,j) \in \mathcal{A}' \cap \mathcal{A}$ for which node i has positive residual capacity. The AF augments as much flow as possible on p while satisfying capacity restrictions for all arcs contained within p and maintaining flow balance at all intermediate nodes visited by p. The AF saturates the flow on an arc with each path augmentation. Alternatively, the m-AF augments as much flow as possible on $\bar{p}$ while satisfying all arc- and node-capacity restrictions and

maintaining flow balance at all intermediate nodes visited by $\bar{p}$. With each augmentation, the m-AF either saturates an arc (i, j) so that $\bar{x}_{ij} = c_{ij}$ or exhausts a node i so that $\sum_{j:(i,j)\in\mathcal{A}} g_{ij}\bar{x}_{ij} = b_i$.

The standard AF terminates with optimal flows for the MFP when no such path p exists. Similarly, when all paths in the residual network originating at s are blocked by a saturated arc or an exhausted node, the m-AF terminates with feasible flows $\bar{x}$; however, these flows need not be optimal to the NCMFP. A full description of the m-AF is given in Appendix A. At the conclusion of this initialization phase, the APC proceeds to the optimality checking phase in Section 3.2.2.

Figure 3.2 displays the residual network after executing the m-AF on the network in Figure 3.1. The solution depicted gives a total flow of $\bar{z} = 8.5$ units. At conclusion of the m-AF, we define

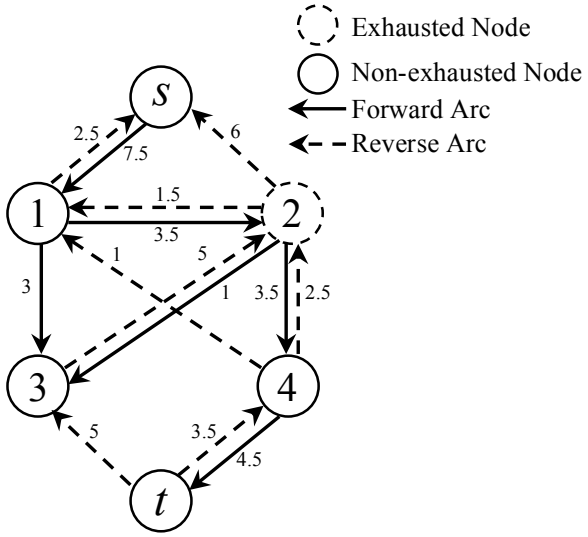


Figure 3.2: Residual network of $\bar{x}$ at m-AF termination

$\bar{b}_i = \sum_{j:(i,j)\in\mathcal{A}} g_{ij}\bar{x}_{ij}$ to be the *residual capacity* of each node $i \in \mathcal{V}$. Defining $\mathcal{V}_R$ as the set of all exhausted nodes, place node i in $\mathcal{V}_R$ when $\bar{b}_i = 0,\ \forall i \in \mathcal{V}$. Next, we define $\bar{c}_{uv} = c_{uv} - \bar{x}_{uv}$ to be the residual capacity of each forward arc $(u, v) \in \mathcal{A}' \cap \mathcal{A}$, $\bar{c}_{uv} = \bar{x}_{vu}$ to be the residual capacity of each reverse arc $(u, v) \in \mathcal{A}' \setminus \mathcal{A}$, and $\bar{z} = \sum_{i:(s,i)\in\mathcal{A}} \bar{x}_{si}$ to be the current maximum flow value.

3.2.2 APC Optimality Checking Phase

We provide a subroutine in this subsection to check if flows $\bar{x}$ are optimal to (3.1) by heuristically seeking a feasible solution to the dual formulation of (3.1) that is complementary slack to $\bar{x}$. If this dual-based heuristic succeeds, then $\bar{x}$ is optimal to (3.1); otherwise, we cannot yet determine if $\bar{x}$ is optimal.

To formulate the dual of (3.1), let variables α_i correspond to the dual values associated with constraints (3.1b), (3.1c), and (3.1d) for all $i \in \mathcal{V}$; let variables γ_{ij} be the dual values for arc-capacity constraints (3.1e) for each $(i,j) \in \mathcal{A}$; and let variables β_i correspond to the dual values for node-capacity constraints (3.1f) for each $i \in \mathcal{V} \setminus \{t\}$. The dual formulation of (3.1) is as follows.

$$\min \sum_{(i,j)\in\mathcal{A}} c_{ij}\gamma_{ij} + \sum_{i\in\mathcal{V}} b_i\beta_i \tag{3.2a}$$

$$\text{s.t. } \alpha_t - \alpha_s = 1 \tag{3.2b}$$

$$\alpha_i - \alpha_j + \gamma_{ij} + g_{ij}\beta_i \geq 0 \quad \forall (i,j) \in \mathcal{A} \tag{3.2c}$$

$$\gamma_{ij} \geq 0 \quad \forall (i,j) \in \mathcal{A} \tag{3.2d}$$

$$\beta_i \geq 0 \quad \forall i \in \mathcal{V} \tag{3.2e}$$

Our dual heuristic algorithm thus seeks to determine a set of α-, β-, and γ-values that satisfy (3.2b)–(3.2e) and the following complementary slackness conditions:

$$\beta_i \left(b_i - \sum_{j:(i,j)\in\mathcal{A}} g_{ij}\bar{x}_{ij} \right) = 0 \quad \forall i \in \mathcal{V} \setminus \{t\}, \tag{3.3a}$$

$$\gamma_{ij}(c_{ij} - \bar{x}_{ij}) = 0 \quad \forall (i,j) \in \mathcal{A}, \tag{3.3b}$$

$$\bar{x}_{ij}(\alpha_i - \alpha_j + \gamma_{ij} + g_{ij}\beta_i) = 0 \quad \forall (i,j) \in \mathcal{A}. \tag{3.3c}$$

Dual Heuristic Initialization: Let $\mathcal{V}_R$ contain all exhausted nodes at the end of the initialization phase (i.e., $\mathcal{V}_R = \{i \in \mathcal{V} : \sum_{j:(i,j)\in\mathcal{A}} g_{ij}\bar{x}_{ij} = b_i\}$). Set $\bar{\mathcal{A}}$ will contain all arcs (i,j) that have been examined by our heuristic in the sense that (3.3c) is satisfied for (i,j). Similarly, $\overline{\mathcal{V}}$ contains the set of examined nodes $i \in \mathcal{V}$ for which a final value for α_i has been chosen. We begin by tentatively setting $\alpha_i = 0$ and $\beta_i = 0,\ \forall i \in \mathcal{V} \setminus \{t\}$, and $\gamma_{ij} = 0,\ \forall (i,j) \in \mathcal{A}$. Initialize $\bar{\mathcal{A}} = \emptyset$ and $\overline{\mathcal{V}} = \{s,t\}$ by setting $\alpha_t = 1$ to satisfy (3.2b). Therefore, (3.3a), (3.3b), and (3.3c) are all satisfied at this point, except for (3.3c) corresponding to all $(i,t) \in \mathcal{A} : \bar{x}_{it} > 0$.

Define $\mathcal{V}_s$ as all nodes $i \in \mathcal{V}$ for which there exists a path $\bar{p}_{si}$ from s to i in $\mathcal{A}'$ that does not visit any nodes in $\mathcal{V}_R$ other than i, which may or may not belong to $\mathcal{V}_R$. For all nodes $i \in \mathcal{V}_s$, keep $\alpha_i = 0$ and place i in $\overline{\mathcal{V}}$. For all arcs $(m,n) \in \mathcal{A}$ contained within each such path $\bar{p}_{si}$, place (m,n) in $\bar{\mathcal{A}}$. Note that for all $(m,n) \in \mathcal{A} : m \in \mathcal{V}_s,\ n \in \mathcal{V}_s$; (3.2c) is satisfied for (m,n) at equality. Define $\mathcal{V}_t$ as all nodes $i \in \mathcal{V}$ for which there exists a path $\bar{p}_{it}$ from i to t in $\mathcal{A}'$ that does not visit

any exhausted nodes (including node i itself). For all nodes $i \in \mathcal{V}_t$, set $\alpha_i = 1$ and place i in $\overline{\mathcal{V}}$. For all arcs $(m, n) \in \mathcal{A}$ contained within each such path $\bar{p}_{it}$, place (m, n) in $\bar{\mathcal{A}}$ since constraint (3.2c) is satisfied at equality for all $(m, n) \in \mathcal{A} : m \in \mathcal{V}_t,\ n \in \mathcal{V}_t$. Both sets $\mathcal{V}_s$ and $\mathcal{V}_t$ can be constructed by executing a breadth-first-search (BFS) in $\mathcal{A}'$ originating at s and t, respectively.

Dual Step 1: For all arcs $(i, j) \in \mathcal{A} \setminus \bar{\mathcal{A}}$ such that $i \in \mathcal{V}_s \setminus \mathcal{V}_R$, $j \in \overline{\mathcal{V}}$, and $\bar{x}_{ij} = c_{ij}$, set $\gamma_{ij} = \alpha_j$ and place (i, j) in $\bar{\mathcal{A}}$. Thus, this choice of γ_{ij} satisfies (3.2c), (3.3b), and (3.3c) for (i, j). If $\bar{\mathcal{A}} = \mathcal{A}$, then go to Dual Step 5. Otherwise, proceed to Dual Step 2.

Dual Step 2: Locate an arc $(\hat{\imath}, \hat{\jmath}) \in \mathcal{A} \setminus \bar{\mathcal{A}} :\ 0 < \bar{x}_{\hat{\imath}\hat{\jmath}} < c_{\hat{\imath}\hat{\jmath}},\ \hat{\imath} \in \mathcal{V}_R \cap \overline{\mathcal{V}}$, and $\hat{\jmath} \in \mathcal{V}_t$. If no such arc exists or if two such arcs $(\hat{\imath}, \hat{\jmath}_1)$ and $(\hat{\imath}, \hat{\jmath}_2)$ exist, then terminate the dual heuristic algorithm without a complementary slack dual-feasible solution. Otherwise, set $\beta_{\hat{\imath}} = (1 - \alpha_{\hat{\imath}})/g_{\hat{\imath}\hat{\jmath}}$ and add arc $(\hat{\imath}, \hat{\jmath})$ to $\bar{\mathcal{A}}$. Note that this choice of $\beta_{\hat{\imath}}$ satisfies (3.2c) and (3.3c) for $(\hat{\imath}, \hat{\jmath})$, and maintains feasibility to (3.3a) for $\hat{\imath}$ because $\hat{\imath} \in \mathcal{V}_R$. Additionally, $\beta_{\hat{\imath}} \geq 0$, since we terminate the dual algorithm if $\alpha_{\hat{\imath}} > 1$ for any $\hat{\imath} \in \mathcal{V}$. Proceed to Dual Step 3.

Dual Step 3: For all arcs $(\hat{\imath}, k) \in \mathcal{A}$ such that $k \in \mathcal{V}_t$ and $\bar{x}_{\hat{\imath}k} = c_{\hat{\imath}k}$, set $\gamma_{\hat{\imath}k} = 1 - \alpha_{\hat{\imath}} - g_{\hat{\imath}k}\beta_{\hat{\imath}}$ and add $(\hat{\imath}, k)$ to $\bar{\mathcal{A}}$. If $\gamma_{\hat{\imath}k} < 0$, then terminate the dual heuristic algorithm without a complementary slack dual-feasible solution. Otherwise, this choice of $\gamma_{\hat{\imath}k}$ satisfies constraints (3.2c) and (3.3c) for $(\hat{\imath}, k)$ without violating (3.3b) since $\bar{x}_{\hat{\imath}k} = c_{\hat{\imath}k}$. Proceed to Dual Step 4a.

Dual Step 4a: Execute a BFS to search for a path $\bar{p}_{\hat{\imath}\hat{m}}$ over arcs in $\mathcal{A}$ originating at $\hat{\imath}$ and terminating at a node $\hat{m} \in \mathcal{V}$ for which there exists an arc $(\hat{m}, \hat{n}) \in \mathcal{A} \setminus \bar{\mathcal{A}} :\ 0 < \bar{x}_{\hat{m}\hat{n}} < c_{\hat{m}\hat{n}}$ and $\hat{n} \in \mathcal{V}$. If no such path exists, return to Dual Step 1. Otherwise, let $\hat{k}$ be the second node visited by $\bar{p}_{\hat{\imath}\hat{m}}$ and proceed to Dual Step 4b.

Dual Step 4b: Execute a BFS to search for a path $\bar{p}_{\hat{n}\hat{u}}$ over arcs in $\mathcal{A}$ originating at $\hat{n}$ and terminating at a node $\hat{u} \in \mathcal{V}$ for which $\bar{x}_{\hat{u}v} = c_{\hat{u}v},\ \forall v \in \mathcal{V}_t : (\hat{u}, v) \in \mathcal{A} \setminus \bar{\mathcal{A}}$. If a path $\bar{p}_{\hat{n}\hat{u}}$ does not exist, then proceed to Dual Step 4c. Otherwise, construct a path $\bar{p}_{\hat{\imath}\hat{u}}$ by joining path $\bar{p}_{\hat{\imath}\hat{m}}$, arc $(\hat{m}, \hat{n})$, and path $\bar{p}_{\hat{n}\hat{u}}$. If $\alpha_j \neq \alpha_{\hat{\imath}} + g_{\hat{\imath}\hat{k}}\beta_{\hat{\imath}}$ for any $j \in \overline{\mathcal{V}}$ visited by $\bar{p}_{\hat{\imath}\hat{u}}$, then terminate the dual heuristic algorithm without a complementary slack dual-feasible solution. Otherwise, for each node $\ell \in \mathcal{V} \setminus \overline{\mathcal{V}}$ visited by $\bar{p}_{\hat{\imath}\hat{u}}$, place ℓ in $\overline{\mathcal{V}}$ and set $\alpha_\ell = \alpha_{\hat{\imath}} + g_{\hat{\imath}\hat{k}}\beta_{\hat{\imath}}$. If $\alpha_\ell > 1$ for any ℓ in $\overline{\mathcal{V}}$, then terminate the

dual heuristic algorithm without a complementary slack dual-feasible solution. Otherwise, for all $v \in \mathcal{V}_t : (\hat{u}, v) \in \mathcal{A} \setminus \bar{\mathcal{A}}$, set $\gamma_{\hat{u}v} = 1 - \alpha_{\hat{\imath}} - g_{\hat{\imath}\hat{k}}\beta_{\hat{\imath}}$ and place $(\hat{u}, v)$ in $\bar{\mathcal{A}}$. If $\gamma_{\hat{u}v} < 0$, then terminate the dual heuristic algorithm without a complementary slack dual-feasible solution. Otherwise, this choice of $\gamma_{\hat{u}v}$ satisfies (3.2c) and (3.3c). Finally, for all arcs $(m, n) \in \mathcal{A} \setminus \bar{\mathcal{A}}$ contained within path $\bar{p}_{\hat{\imath}\hat{u}}$, place (m, n) in $\bar{\mathcal{A}}$ since our choice of α_m and α_n satisfies (3.2c) and (3.3c). Return to Dual Step 4a.

Dual Step 4c: Execute a BFS to search for a path $\bar{p}_{\hat{n}\hat{u}}$ over arcs in $\mathcal{A}$ originating at $\hat{n}$ and terminating at a node $\hat{u} \in \mathcal{V}_R \setminus \overline{\mathcal{V}}$. If no such path $\bar{p}_{\hat{n}\hat{u}}$ exists, then return to Dual Step 1. Otherwise, construct a path $\bar{p}_{\hat{\imath}\hat{u}}$ by joining path $\bar{p}_{\hat{\imath}\hat{m}}$, arc $(\hat{m}, \hat{n})$, and path $\bar{p}_{\hat{n}\hat{u}}$. If $\alpha_j \neq \alpha_{\hat{\imath}} + g_{\hat{\imath}\hat{k}}\beta_{\hat{\imath}}$ for any $j \in \overline{\mathcal{V}}$ visited by $\bar{p}_{\hat{\imath}\hat{u}}$, then terminate the dual heuristic algorithm without a complementary slack dual-feasible solution. Otherwise, for each node $\ell \in \mathcal{V} \setminus \overline{\mathcal{V}}$ visited by $\bar{p}_{\hat{\imath}\hat{u}}$, place ℓ in $\overline{\mathcal{V}}$ and set $\alpha_\ell = \alpha_{\hat{\imath}} + g_{\hat{\imath}\hat{k}}\beta_{\hat{\imath}}$. If $\alpha_\ell > 1$ for any ℓ in $\overline{\mathcal{V}}$, then terminate the dual heuristic algorithm without a complementary slack dual-feasible solution. Otherwise, for all arcs (m, n) contained within path $\bar{p}_{\hat{\imath}\hat{u}}$, place (m, n) in $\bar{\mathcal{A}}$ since our choice of α_m and α_n satisfies (3.2c) and (3.3c). Return to Dual Step 4a.

Dual Step 5: Terminate successfully with a set of dual values complementary slack to $\bar{x}$. Thus, flows $\bar{x}$ are optimal to (3.1).

If this heuristic algorithm does not determine that flows $\bar{x}$ are optimal, then the APC proceeds to the augmenting-flow phase of the APC in Section 3.2.3 to either prove the optimality of $\bar{x}$ or to identify a set of flow adjustments that allow for additional flow to be augmented on an s–t path without violating any constraints (3.1b)–(3.1f).

We illustrate our dual heuristic algorithm on the network of flows $\bar{x}$ in Figure 3.3.

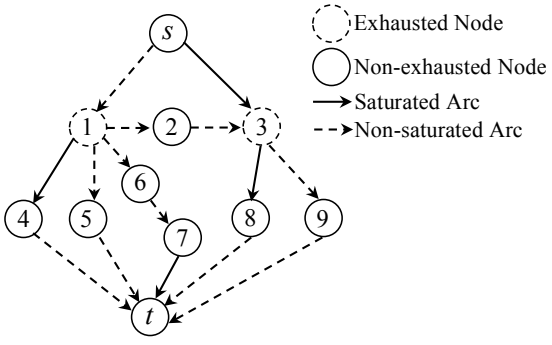


Figure 3.3: Dual heuristic example

Dual Initialization: Initialize by setting all dual values equal to 0, $\mathcal{V}_R = \{1, 3\}$, $\mathcal{V}_s = \{s, 1\}$, and $\mathcal{V}_t = \{4, 5, 8, 9, t\}$. Next, set $\alpha_i = 1,\ \forall i \in \mathcal{V}_t$, and place all nodes $i \in \mathcal{V}_s \cup \mathcal{V}_t$ in $\overline{\mathcal{V}}$. Finally, place arcs $(s, 1),\ (4, t),\ (5, t),\ (8, t)$ and $(9, t)$ in $\bar{\mathcal{A}}$. For this network of flows, no arcs $(i, j) \in \mathcal{A}$ exist for which $i \in \mathcal{V}_s,\ j \in \mathcal{V}_t$, and $\bar{x}_{ij} = c_{ij}$.

Dual Step 1: There exist no arcs $(i, j) \in \mathcal{A} \setminus \bar{\mathcal{A}}$ for which $i \in \mathcal{V}_s \setminus \mathcal{V}_R$, $j \in \overline{\mathcal{V}}$, and $\bar{x}_{ij} = c_{ij}$. Proceed to Dual Step 2.

Dual Step 2: Set $(\hat{\imath}, \hat{\jmath}) = (1, 5)$ since node $1 \in \overline{\mathcal{V}} \cap \mathcal{V}_R$ and node $5 \in \mathcal{V}_t$. Set $\beta_1 = 1/g_{15}$, place $(1, 5)$ in $\bar{\mathcal{A}}$, and proceed to Dual Step 3.

Dual Step 3: Set $\gamma_{14} = 1 - g_{14}\beta_1$ because $\bar{x}_{14} = c_{14}$ and node $4 \in \mathcal{V}_t$. Place $(1, 4)$ in $\bar{\mathcal{A}}$, and proceed to Dual Step 4a.

Dual Step 4a: Since $(1, 6) \notin \bar{\mathcal{A}}$ originates at 1 and $0 < \bar{x}_{16} < c_{16}$, then set $\hat{m} = 1$, $\hat{n} = 6$ and $\hat{k} = 6$. Proceed to Dual Step 4b.

Dual Step 4b: Since $\bar{x}_{7t} = c_{7t}$ and $t \in \mathcal{V}_t$, then set $\hat{u} = 7$ and $\bar{p}_{\hat{n}\hat{u}} = (6, 7)$. Construct path $\bar{p}_{\hat{\imath}\hat{u}} = (1, 6, 7)$ by joining arcs $(1, 6)$ and $(6, 7)$. Thus, set $\alpha_6 = \alpha_7 = g_{16}\beta_1$ and add nodes 6 and 7 to $\overline{\mathcal{V}}$. Next, set $\gamma_{7t} = 1 - g_{16}\beta_1$ and add arcs $(1, 6),\ (6, 7)$, and $(7, t)$ to $\bar{\mathcal{A}}$. Return to Dual Step 4a.

Dual Step 4a: Since $(1, 2) \notin \bar{\mathcal{A}}$ originates at 1 and $0 < \bar{x}_{12} < c_{12}$, then set $\hat{m} = 1$, $\hat{n} = 2$, and $\hat{k} = 2$. Proceed to Dual Step 4b.

Dual Step 4b: No such path exists. Proceed to Dual Step 4c.

Dual Step 4c: Since $3 \in \mathcal{V}_R$, then set $\hat{u} = 3$ and $\bar{p}_{\hat{n}\hat{u}} = (2, 3)$. Construct path $\bar{p}_{\hat{\imath}\hat{u}} = (1, 2, 3)$ by joining arcs $(1, 2)$ and $(2, 3)$. Thus, set $\alpha_2 = \alpha_3 = g_{12}\beta_1$, add nodes 2 and 3 to $\overline{\mathcal{V}}$, and add arcs $(1, 2)$ and $(2, 3)$ to $\bar{\mathcal{A}}$. Repeat Dual Step 4a.

Dual Step 4a: No such path exists. Return to Dual Step 1.

Dual Step 1: Set $\gamma_{s3} = \alpha_3$ because $s \in \mathcal{V}_s \setminus \mathcal{V}_R$, $3 \in \overline{\mathcal{V}}$, and $\bar{x}_{s3} = c_{s3}$. Then, add arc $(s, 3)$ to $\bar{\mathcal{A}}$ and proceed to Dual Step 2.

Dual Step 2: Set $(\hat{\imath}, \hat{\jmath}) = (3, 9)$ because $3 \in \overline{\mathcal{V}} \cap \mathcal{V}_R$, node $9 \in \mathcal{V}_t$, and $(3, 9) \in \mathcal{A} \setminus \bar{\mathcal{A}}$. Set $\beta_3 = (1 - \alpha_3)/g_{39}$, place $(3, 9)$ in $\bar{\mathcal{A}}$, and proceed to Dual Step 3.

Dual Step 3: Since $\bar{x}_{38} = c_{38}$ and node $8 \in \mathcal{V}_t$, set $\gamma_{38} = 1 - \alpha_3 - g_{38}\beta_3$, place $(3, 8)$ in $\bar{\mathcal{A}}$, and proceed to Dual Step 4a.

Dual Step 4a: No such path exists. Return to Dual Step 1.

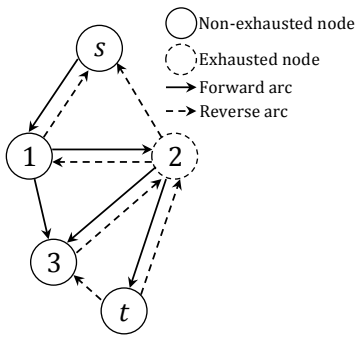
Dual Step 1: Since $\bar{\mathcal{A}} = \mathcal{A}$, then go to Dual Step 5.

Dual Step 5: Terminate successfully with a set of dual values complementary slack to $\bar{x}$. Thus, flows $\bar{x}$ are optimal.

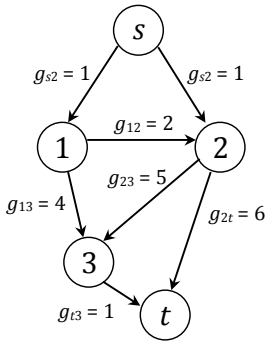
3.2.3 APC Augmenting-flow Phase

At this point in the algorithm, if flows $\bar{x}$ have yet to be deemed optimal in Section 3.2.2, then all s–t paths in $\mathcal{A}'$ must visit a node in $\mathcal{V}_R$. Recalling that $\bar{b}_i = 0$ for all $i \in \mathcal{V}_R$, we identify a strategy for adjusting flows $\bar{x}$ in such a way that increases $\bar{b}_i$ for some nonempty set of nodes $i \in \mathcal{V}_R$ without violating the node- or arc-capacity restriction or flow balance at any node $i \in \mathcal{V}$ in order to increase the current maximum flow value $\bar{z}$. These flow adjustments thus correspond to sending flows across a set of augmenting cycles in $\mathcal{A}'$.

Consider the following example residual network $\mathcal{A}'$ corresponding to flows $\bar{x}$ in Figure 3.4 that details an example of when this set contains a single augmenting cycle.



(a) Residual network corresponding to flows $\bar{x}$



(b) Network displaying g-values

Figure 3.4: Flow adjusting circulation example

Without adjusting flows in Figure 3.4a, additional flow cannot be sent from s to t since the only s–t path $(s, 1, 2, t)$ visits exhausted node 2. Note that adjusting flows on the directed cycle $(1, 3, 2, 1)$ by a single unit of flow increases the residual capacity of node 2 by $g_{23} = 5$ without decreasing the residual capacity of any other exhausted node to allow for at most 5/6 units of flow to be augmented on $(s, 1, 2, t)$. Any augmenting cycle that increases the residual capacity of a node $i \in \mathcal{V}_R$ without decreasing the residual capacity of any node in $\mathcal{V}_R \setminus \{i\}$ is referred to as a *capacity-increasing cycle*.

Additionally, a set of multiple augmenting cycles may exist that are not individually capacity-increasing but jointly increase the residual capacity of a set of nodes in $\mathcal{V}_R$ without decreasing the residual capacity of any other nodes in $\mathcal{V}_R$. We refer to such set as a *joint-cycle set*. As an example, consider the following residual network in Figure 3.5. For this example, let $g_{12} = 2$, $g_{13} = 9$, $g_{14} = 10$, $g_{23} = 5$, and $g_{24} = 6$, and observe that $\mathcal{V}_R = \{1, 2\}$. Consider cycle 1, given by $(1, 2, 3, 1)$, and cycle

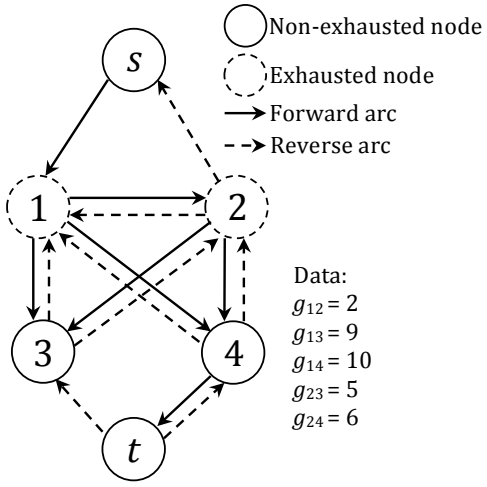


Figure 3.5: Residual network motivating the LP phase

2, given by $(1, 4, 2, 1)$. Adjusting flow on cycle 1 increases the residual capacity of node 1 at a rate of $g_{13} - g_{12} = 7$ but decreases the residual capacity of node 2 at a rate of $g_{23} = 5$, while adjusting flow on cycle 2 increases the residual capacity of node 2 at a rate of $g_{24} = 6$ but decreases the residual capacity of node 1 at a rate of $g_{14} - g_{12} = 8$. Thus, neither cycle is capacity-increasing.

However, a combination of flow adjustments on these two cycles may simultaneously increase the residual capacity of both nodes 1 and 2. Letting the flow on cycles 1 and 2 be Δ_1 and Δ_2,

respectively, the residual capacity of nodes 1 and 2 are given as:

$$\bar{b}_1 = (g_{13} - g_{12})\Delta_1 - (g_{14} - g_{12})\Delta_2 = 7\Delta_1 - 8\Delta_2,$$
$$\bar{b}_2 = -g_{23}\Delta_1 + g_{24}\Delta_2 = -5\Delta_1 + 6\Delta_2.$$

By setting $\Delta_1 = 1$ and $\Delta_2 = 21/25$, the residual capacities $\bar{b}_1$ and $\bar{b}_2$ increase by $7/25$ and $1/25$, respectively, to allow for additional flow to be augmented on $(s, 1, 2, 4, t)$.

For some cases, the process of identifying a set of capacity-increasing cycles that allow for more flow to be sent from s to t can be accomplished without solving an LP, as we will show in Section 3.3. Alternatively, detecting the existence of a joint-cycle set is evidently more difficult, and so we resort to the use of linear programming to either find a joint-cycle set among arcs in $\mathcal{A}'$ or prove that no joint-cycle set exists. Defining the residual arc capacity $\bar{c}$ as before, let continuous variables $y_{ij} \geq 0,\ \forall (i,j) \in \mathcal{A}' : \bar{c}_{ij} > 0$, be positive when arc (i,j) is contained within an augmenting cycle found within a joint-cycle set. Let continuous variables $0 \leq s_i \leq 1,\ \forall i \in \mathcal{V}_R$, be positive when adjusting flows $\bar{x}$ according to y increases residual capacity at node i, and 0 otherwise. The following LP seeks to find a joint-cycle set that maximizes the number of nodes in $\mathcal{V}_R$ whose residual capacity can be increased by adjusting flows according to y.

$$\max \sum_{i \in \mathcal{V}_R} s_i \tag{3.4a}$$

$$\text{s.t.} \sum_{j:(i,j)\in\mathcal{A}',\ \bar{c}_{ij}>0} y_{ij} - \sum_{k:(k,i)\in\mathcal{A}',\ \bar{c}_{ki}>0} y_{ki} = 0 \quad \forall i \in \mathcal{V} \tag{3.4b}$$

$$\sum_{k:(k,i)\in\mathcal{A}'\setminus\mathcal{A},\ \bar{c}_{ki}>0} g_{ik} y_{ki} - \sum_{j:(i,j)\in\mathcal{A}'\cap\mathcal{A},\ \bar{c}_{ij}>0} g_{ij} y_{ij} - s_i \geq 0 \quad \forall i \in \mathcal{V}_R \tag{3.4c}$$

$$0 \leq s_i \leq 1 \quad \forall i \in \mathcal{V}_R \tag{3.4d}$$

$$y_{ij} \geq 0 \quad \forall (i,j) \in \mathcal{A}' : \bar{c}_{ij} > 0 \tag{3.4e}$$

Objective function (3.4a) maximizes the number of nodes in $\mathcal{V}_R$ whose residual capacity can be increased, and constraints (3.4b) ensure that flows y, if not all zero, form a set of augmenting cycles that comprise a joint-cycle set. Constraints (3.4c) and (3.4d) ensure variable s_i equals 1 when the residual capacity at node $i \in \mathcal{V}_R$ increases as a result of adjusting flows according to y, as proved in Proposition 1 below. Finally, constraints (3.4e) ensure non-negativity.

Proposition 1. *Variables $s_i,\ \forall i \in \mathcal{V}_R$, are binary in any optimal solution to* (3.4).

Proof. Let $\bar{s}$ and $\bar{y}$ be the vector of s-values and y-values, respectively, for a given optimal solution to (3.4), having an objective function value $s^* = \sum_{i \in \mathcal{V}_R} \bar{s}_i$. Assume by contradiction that $0 < \bar{s}_i < 1$ for some node $i \in \mathcal{V}_R$. We can modify this solution by multiplying $\bar{y}$ by scalar $1/\bar{s}_i$ and setting $\bar{s}_i = 1$ with all other $\bar{s}$-values remaining constant. This solution remains feasible to constraints (3.4b)–(3.4e) since $1/\bar{s}_i > 1$. Additionally, s^* increases by $1 - \bar{s}_i$, which contradicts the optimality of the original solution. This completes the proof. □

The augmenting-flow phase of the APC thus proceeds as follows. Let Θ be the index set of all augmenting cycles found so far, and let $\bar{y}$ be the set of flows in an optimal solution to (3.4). We identify capacity-increasing cycles by executing a DFS originating at each node $i \in \mathcal{V}_R$, for which $s_i = 1$ in an optimal solution to (3.4), among those arcs $(i, j) \in \mathcal{A}' : \bar{y}_{ij} > 0$. Next, place each identified cycle in Θ, and remove any duplicate cycles found in Θ. Defining $\overline{\mathcal{V}}_R$ as the set of all nodes $i \in \mathcal{V}_R$ for which the APC has found an augmenting cycle that increases the residual capacity at node i, place node i in $\overline{\mathcal{V}}_R$ if $s_i = 1$ in an optimal solution to (3.4). Determining the cycles in Θ in this way ensures that Θ consists of at least one cycle that increases the residual capacity at each node in $\overline{\mathcal{V}}_R$. Next, execute a DFS to search for an s–t path $\bar{p}$ among those arcs in $\mathcal{A}'$ that visits only those nodes in $\{\mathcal{V} \setminus \mathcal{V}_R\} \cup \{\overline{\mathcal{V}}_R\}$. If no path $\bar{p}$ exists, then each s–t path p in $\mathcal{A}'$ visits some set of exhausted nodes in $\mathcal{V}$, and no capacity-increasing cycles exist that jointly increase the capacity of all exhausted nodes visited by p. In this case, flows $\bar{x}$ are deemed optimal according to the proof of optimality detailed in Section 3.2.4.

When some path $\bar{p}$ exists, decomposing flows $\bar{y}$, from an optimal (3.4) solution, into the set of augmenting cycles Θ in the aforementioned way typically allows for additional flow to be sent on $\bar{p}$. When augmenting flow on cycles in Θ does not allow for positive flow to be sent on $\bar{p}$, there exists some alternative decomposition of flows $\bar{y}$ into a set of capacity-increasing cycles that does allow for flow to be augmented on $\bar{p}$. For such cases, the current solution obtained by the APC is not yet optimal. Determining this alternative flow decomposition is non-trivial and could require solving an additional LP larger than either (3.4) or (3.5). Thus, we terminate the APC with flows $\bar{x}$ feasible to the NCMFP when such cases occur.

Thus, we must determine the amount by which to adjust flow on each cycle in Θ to maximize the allowable flow on $\bar{p}$. To do so, define $\mathcal{A}_{\bar{p}}$ as the set of all arcs contained within $\bar{p}$ and $\mathcal{V}_{\bar{p}}$ as the set of all nodes visited by $\bar{p}$, and let λ represent the amount by which we augment flow on $\bar{p}$. Next, remove any cycles in Θ that do not visit a node in $\mathcal{V}_{\bar{p}}$, let Δ_k represent the amount by which we adjust flow on each cycle $\theta_k \in \Theta$, and let θ_k be the set of all arcs contained within each cycle $\theta_k \in \Theta$.

Let set $\mathcal{V}_R^{\bar{p}} \subseteq \mathcal{V}_R$ consist of all exhausted nodes visited by $\bar{p}$, let $\mathcal{V}_\Theta$ be the set of all nodes visited by a cycle in Θ, and let $\mathcal{A}_\Theta$ be the set of all arcs contained within a cycle in Θ. Define $\Theta_i \subseteq \Theta$ to be the set of all cycles visiting node $i \in \mathcal{V}_\Theta$, and let set $\Theta_{uv} \subseteq \Theta$ consist of all cycles containing arc $(u,v) \in \mathcal{A}_\Theta$. Finally, define residual capacities $\bar{b}_i = \sum_{j:(i,j)\in\mathcal{A}} g_{ij}\bar{x}_{ij},\ \forall i \in \mathcal{V}$; $\bar{c}_{uv} = c_{uv} - \bar{x}_{uv},\ \forall (u,v) \in \mathcal{A}' \cap \mathcal{A}$; and $\bar{c}_{vu} = \bar{x}_{uv},\ \forall (v,u) \in \mathcal{A}' \setminus \mathcal{A}$.

The following LP can now be solved to determine a set of flow adjustments $\Delta_k,\ \forall \theta_k \in \Theta$, that allow for a maximum amount of flow λ to be augmented on path $\bar{p}$ while maintaining the feasibility of flows $\bar{x}$.

$$\max\ \lambda \tag{3.5a}$$

$$\text{s.t.} \sum_{\theta_k \in \Theta_i} \left(\sum_{j:(i,j)\in\mathcal{A}^k\cap\{\mathcal{A}'\cap\mathcal{A}\}} g_{ij}\Delta_k - \sum_{h:(h,i)\in\mathcal{A}^k\cap\{\mathcal{A}'\setminus\mathcal{A}\}} g_{ih}\Delta_k \right) + \sum_{m:(i,m)\in\mathcal{A}_{\bar{p}}\cap\{\mathcal{A}'\cap\mathcal{A}\}} g_{im}\lambda - \sum_{\ell:(\ell,i)\in\mathcal{A}_{\bar{p}}\cap\{\mathcal{A}'\setminus\mathcal{A}\}} g_{i\ell}\lambda \le 0,\ \forall i \in \mathcal{V}_R^{\bar{p}}, \tag{3.5b}$$

$$\sum_{\theta_k \in \Theta_i} \left(\sum_{j:(i,j)\in\mathcal{A}^k\cap\{\mathcal{A}'\cap\mathcal{A}\}} g_{ij}\Delta_k - \sum_{h:(h,i)\in\mathcal{A}^k\cap\{\mathcal{A}'\setminus\mathcal{A}\}} g_{ih}\Delta_k \right) + \sum_{m:(i,m)\in\mathcal{A}_{\bar{p}}\cap\{\mathcal{A}'\cap\mathcal{A}\}} g_{im}\lambda - \sum_{\ell:(\ell,i)\in\mathcal{A}_{\bar{p}}\cap\{\mathcal{A}'\setminus\mathcal{A}\}} g_{i\ell}\lambda \le \bar{b}_i,\ \forall i \in \mathcal{V}_\Theta \setminus \mathcal{V}_R^{\bar{p}}, \tag{3.5c}$$

$$\sum_{m:(i,m)\in\mathcal{A}_{\bar{p}}\cap\{\mathcal{A}'\cap\mathcal{A}\}} g_{im}\lambda - \sum_{\ell:(\ell,i)\in\mathcal{A}_{\bar{p}}\cap\{\mathcal{A}'\setminus\mathcal{A}\}} g_{i\ell}\lambda \le \bar{b}_i,\ \forall i \in \mathcal{V}_{\bar{p}} \setminus \mathcal{V}_R^{\bar{p}}, \tag{3.5d}$$

$$-\bar{c}_{vu} \le \sum_{\theta_k\in\Theta_{uv}} \Delta_k - \sum_{\theta_m\in\Theta_{vu}} \Delta_m + \lambda \le \bar{c}_{uv} \quad \forall (u,v) \in \mathcal{A}' \cap \mathcal{A}_{\bar{p}} \cap \mathcal{A}_\Theta, \tag{3.5e}$$

$$-\bar{c}_{vu} \le \sum_{\theta_k\in\Theta_{uv}} \Delta_k - \sum_{\theta_m\in\Theta_{vu}} \Delta_m \le \bar{c}_{uv} \quad \forall (u,v) \in \{\mathcal{A}' \cap \mathcal{A}_\Theta\} \setminus \mathcal{A}_{\bar{p}}, \tag{3.5f}$$

$$-\bar{c}_{vu} \le \lambda \le \bar{c}_{uv} \quad \forall (u,v) \in \{\mathcal{A}' \cap \mathcal{A}_{\bar{p}}\} \setminus \mathcal{A}_\Theta, \tag{3.5g}$$

$$\Delta_k \ge 0,\ \forall \theta_k \in \Theta, \tag{3.5h}$$

$$\lambda \ge 0. \tag{3.5i}$$

Objective function (3.5a) maximizes the flow λ on $\bar{p}$, while constraints (3.5b) ensure that capacity consumed by adjusting flow at each exhausted node visited by $\bar{p}$ is non-positive. Constraints (3.5c) similarly ensure that the capacity consumed by adjusting flow at each node visited by a cycle in Θ but not $\bar{p}$ does not exceed residual capacity $\bar{b}_i$ at node i. Constraints (3.5d) ensure that the capacity consumed by adjusting flow at each node visited by $\bar{p}$ but not any cycle in Θ does not exceed

residual capacity $\bar{b}_i$ at node i. Constraints (3.5e) guarantee that flow adjustments do not violate the non-negativity or arc-capacity restrictions on each arc (u, v) contained within $\bar{p}$ and a cycle in Θ. Constraints (3.5f) similarly guarantee that the flow on each arc contained within a cycle in Θ but not within $\bar{p}$ remains non-negative and does not violate arc-capacity restrictions. Constraints (3.5g) guarantee that the flow on each arc (u, v) contained within $\bar{p}$ but not within a cycle in Θ remains non-negative and does not violate arc-capacity restrictions. Finally, (3.5h) and (3.5i) enforce non-negative flows on each cycle in Θ and on path $\bar{p}$, respectively.

An optimal solution to (3.5) provides a set of flow adjustments $\bar{\Delta}_k,\ \forall \theta_k \in \Theta$, and the flow $\bar{\lambda}$ to be augmented path $\bar{p}$. Thus, to conclude this phase of the APC, solve model (3.5). If $\bar{\lambda} = 0$, then there exists an alternative decomposition of flows $\bar{y}$ that allows for positive flow to be augmented on $\bar{p}$, and we terminate the APC with feasible flows $\bar{x}$.

Otherwise, if $\bar{\lambda} > 0$, then update flows $\bar{x}$ by setting

$$\bar{x}_{uv} = \bar{x}_{uv} + \bar{\lambda} + \sum_{\theta_k \in \Theta_{uv}} \bar{\Delta}_k - \sum_{\theta_k \in \Theta_{vu}} \bar{\Delta}_k,\ \forall (u,v) \in \mathcal{A}_{\bar{p}} \cap \{\mathcal{A}' \cap \mathcal{A}\},$$

$$\bar{x}_{uv} = \bar{x}_{uv} - \bar{\lambda} + \sum_{\theta_k \in \Theta_{uv}} \bar{\Delta}_k - \sum_{\theta_k \in \Theta_{vu}} \bar{\Delta}_k,\ \forall (v,u) \in \mathcal{A}_{\bar{p}} \cap \{\mathcal{A}' \setminus \mathcal{A}\},$$

$$\bar{x}_{uv} = \bar{x}_{uv} + \sum_{\theta_k \in \Theta_{uv}} \bar{\Delta}_k - \sum_{\theta_k \in \Theta_{vu}} \bar{\Delta}_k,\ \forall (u,v) \in \{\mathcal{A}_\Theta \setminus \mathcal{A}_{\bar{p}}\} \cap \{\mathcal{A}' \cap \mathcal{A}\}.$$

For all forward arcs $(u, v) \in \mathcal{A}' \cap \mathcal{A}$, update residual capacity $\bar{c}_{uv} = c_{uv} - \bar{x}_{uv}$. For all reverse arcs $(v, u) \in \mathcal{A}' \setminus \mathcal{A}$, update residual capacity $\bar{c}_{vu} = \bar{x}_{uv}$. Finally, update residual node capacity $\bar{b}_i = b_i - \sum_{j:(i,j)\in\mathcal{A}} g_{ij}\bar{x}_{ij}$, update current maximum flow value $\bar{z} = \bar{z} + \bar{\lambda}$, and return to the optimality checking phase of the APC to check if flows $\bar{x}$ can be deemed optimal.

3.2.4 APC Algorithm Optimality

Theorem 1. *Let $\bar{x}$ be flows determined by the APC algorithm, such that each s–t path p in $\mathcal{A}'$ visits some set of exhausted nodes in $\mathcal{V}_R$, and no capacity-increasing cycles exist that jointly increase the residual capacity of all exhausted nodes visited by p. Flows $\bar{x}$ are optimal to the NCMFP.*

Proof. We prove Theorem 1 by constructing a dual-feasible solution to (3.2) for which the following complementary slackness conditions are satisfied: $\beta_i(b_i - \sum_{j:(i,j)\in\mathcal{A}} g_{ij}\bar{x}_{ij}) = 0,\ \forall i \in \mathcal{V}$; $\gamma_{ij}(c_{ij} - \bar{x}_{ij}) = 0,\ \forall (i,j) \in \mathcal{A}$; and constraints (3.2c) are satisfied at equality for all positive-flow arcs. To provide such a solution, we first define $\beta_i = 0,\ \forall i \in \mathcal{V} \setminus \mathcal{V}_R$; $\gamma_{ij} = 0$, for all arcs $(i,j) \in \mathcal{A} : \bar{x}_{ij} < c_{ij}$ and $\gamma_{ij} = \alpha_j - \alpha_i - g_{ij}\beta_i$ for all $(i,j) \in \mathcal{A} :\ \bar{x}_{ij} = c_{ij}$. The first two complementary slackness conditions are thus satisfied.

We define dual values $\alpha_i,\ \forall i \in \mathcal{V}$, and $\beta_i,\ \forall i \in \mathcal{V}_R$, according to their right-hand shadow price (RHSP) interpretation. That is, let $\mathcal{A}'$ consist of all arcs in the residual network corresponding

to $\bar{x}$. For $i \in \mathcal{V}$, we compute α_i by placing an infinitesimal supply at node i and determining the proportion of that supply that can be sent to t on arcs in $\mathcal{A}'$ without violating node-capacity constraints. The values for β_i, for all $i \in \mathcal{V}_R$, are defined analogously when an additional unit of capacity is allocated to node i. More precisely, the following LP models define $\alpha_i,\ \forall i \in \mathcal{V}$, and $\beta_i,\ \forall i \in \mathcal{V}_R$.

Model (3.6) computes α_i for node $i \in \mathcal{V}$ by determining a set of flows y^i that maximize flow f_i into node t when an additional unit of supply originates at node i.

$$\max\ f_i \tag{3.6a}$$

$$\text{s.t.} \sum_{j:(j,t)\in\mathcal{A}'} y_{jt} = f_i \tag{3.6b}$$

$$\sum_{j:(i,k)\in\mathcal{A}'} y_{ik} - \sum_{h:(h,i)\in\mathcal{A}'} y_{hi} = 1 \tag{3.6c}$$

$$\sum_{w:(v,w)\in\mathcal{A}'} y_{vw} - \sum_{u:(u,v)\in\mathcal{A}'} y_{uv} = 0 \quad \forall v \in \mathcal{V}\setminus\{s,i,t\} \tag{3.6d}$$

$$\sum_{v:(u,v)\in\mathcal{A}'\cap\mathcal{A}} g_{uv}y_{uv} - \sum_{v:(u,v)\in\mathcal{A}'\setminus\mathcal{A}} g_{vu}y_{uv} \le 0 \quad \forall u \in \mathcal{V}_R \tag{3.6e}$$

$$y_{uv} \ge 0 \quad \forall (u,v) \in \mathcal{A}' \tag{3.6f}$$

Objective function (3.6a) and constraint (3.6b) maximize flow into t. Constraint (3.6c) ensures that a single unit of flow originates at i, and constraints (3.6d) ensure flow balance at all nodes in $\mathcal{V}\setminus\{s,i,t\}$. Constraints (3.6e) ensure that the net capacity consumption at each node $u \in \mathcal{V}_R$ is less than or equal to 0, noting that arcs in $\mathcal{A}\cap\mathcal{A}'$ refer to forward arcs in the residual network and $\mathcal{A}'\setminus\mathcal{A}$ refers to backward arcs in the residual network. Constraints (3.6f) state non-negativity restrictions on the flow variables. We set α_i equal to the optimal objective function value of (3.6).

Similar to (3.6), model (3.7) computes β_i for each node $i \in \mathcal{V}_R$ by determining a set of flows $\bar{y}^i$ for which flow $\bar{f}_i$ into t is maximized when one unit of capacity is allocated to node i.

$$\max\ \bar{f}_i \tag{3.7a}$$

$$\text{s.t.} \sum_{v:(v,t)\in\mathcal{A}'} \bar{y}_{vt} = \bar{f}_i \tag{3.7b}$$

$$\sum_{w:(v,w)\in\mathcal{A}'} \bar{y}_{vw} - \sum_{u:(u,v)\in\mathcal{A}'} \bar{y}_{uv} = 0 \quad \forall v \in \mathcal{V}\setminus\{s,t\} \tag{3.7c}$$

$$\sum_{k:(i,k)\in\mathcal{A}'\cap\mathcal{A}} g_{ik}\bar{y}_{ik} - \sum_{k:(i,k)\in\mathcal{A}'\setminus\mathcal{A}} g_{ki}\bar{y}_{ik} \le 1 \tag{3.7d}$$

$$\sum_{w:(v,w)\in\mathcal{A}'\cap\mathcal{A}} g_{vw}\bar{y}_{vw} - \sum_{w:(v,w)\in\mathcal{A}'\setminus\mathcal{A}} g_{wv}\bar{y}_{vw} \le 0 \quad \forall v \in \mathcal{V}_R\setminus\{i\} \tag{3.7e}$$

$$\bar{y}_{uv} \ge 0 \quad \forall (u,v) \in \mathcal{A}' \tag{3.7f}$$

Constraints (3.7d) and (3.7e) ensure that the net node capacity consumption at $i \in \mathcal{V}_R$ is less than or equal to 1 and 0 at all nodes in $\mathcal{V}_R\setminus\{i\}$. We set β_i equal to the optimal objective value of (3.7).

We now show that our duals satisfy the constraints of (3.2), and that arc-capacity restrictions (3.2c) written for $(i,j) \in \mathcal{A}$ holds as an equality when $\bar{x}_{ij} > 0$. To accomplish this, we examine (3.2c)

for some $(i,j) \in \mathcal{A}$ by considering three possible cases: (*a*) $\bar{x}_{ij} = 0$, (*b*) $\bar{x}_{ij} = c_{ij}$, or (*c*) $0 < \bar{x}_{ij} < c_{ij}$. Let y^{i*} and $\bar{y}^{i*}$ be optimal solutions to models (3.6) and (3.7) corresponding to a node $i \in \mathcal{V}$, having objective function values α_i^* and β_i^*, respectively.

For case (*a*), the fact that $\bar{x}_{ij} = 0$ implies that $\gamma_{ij} = 0$, and so we show that $\alpha_i^* + g_{ij}\beta_i^* \geq \alpha_j^*$. By contradiction, suppose that $\alpha_j^* - \alpha_i^* - g_{ij}\beta_i^* > 0$. Define $y' = y^{i*} + \bar{y}^{i*}$ as the aggregation solution that results from simultaneously incrementing one unit of supply at node i and one unit of capacity at node i. Define ε to be a small positive number and e_{ij} to be a column vector of size $|\mathcal{A}'|$ with a 1 in the element corresponding to (i,j) and zeros elsewhere. Consider the alternative aggregate flow solution $y'' = (1-\varepsilon)y^{i*} + (1 - g_{ij}\varepsilon)\bar{y}^{i*} + \varepsilon(e_{ij} + y^{j*})$. We claim that y'' contains the flows for which the net node-i capacity consumption increases by no more than 1, and the net flow out of node i increases by 1. To see that, let κ_i be the left-hand-side value of (3.6e), written for node i, corresponding to y^{i*}, and let $\bar{\kappa}_i$ be the left-hand-side value of (3.6e) corresponding to $\bar{y}^{i*}$ (where $\kappa_i = 0$ if $i \notin \mathcal{V}_R$). Note that $\kappa_i \leq 0$ and $\bar{k}_i \leq 1$. In y'', the net node-i capacity consumption becomes $(1-\varepsilon)\kappa_i + (1 - g_{ij}\varepsilon)\bar{\kappa}_i + \varepsilon g_{ij} \leq 1$, with equality holding only when $\kappa_i = 0$ and $\bar{\kappa}_i = 0$. Also, the net flow out of node i is given by $(1-\varepsilon) + 0 + \varepsilon = 1$ in solution y''. Observe that y'' decomposes into two feasible solutions to (3.6) and (3.7) written for node i, and that the combined objective of y''(i.e., the flow reaching node t) is $(1-\varepsilon)\alpha_i^* + (1 - g_{ij}\varepsilon)\beta_i^* + \varepsilon\alpha_j^* = \alpha_i^* + \beta_i^* + \varepsilon(\alpha_j^* - \alpha_i^* - g_{ij}\beta_i^*)$. That objective exceeds the combined objective of y', $\alpha_i^* + \beta_i^*$, by assumption that $\alpha_j^* - \alpha_i^* - g_{ij}\beta_i^* > 0$, contradicting the optimality of y^{i*} and $\bar{y}^{i*}$.

For case (*b*), because we have that $\gamma_{ij} = \alpha_j^* - \alpha_i^* - g_{ij}\beta_i^*$, we only need to show that $\alpha_i^* + g_{ij}\beta_i^* \leq \alpha_j^*$. By contradiction, suppose that $\alpha_j^* < \alpha_i^* + g_{ij}\beta_i^*$. Since there exists positive capacity on arc $(j,i) \in \mathcal{A}' \setminus \mathcal{A}$, we can employ a similar strategy as with case (*a*) by exchanging indices i and j to build an alternative solution to (3.6) for node j having an objective value equal to $\alpha_i^* + \beta_i^* + \varepsilon(\alpha_i^* + g_{ij}\beta_i^* - \alpha_j^*)$. This objective exceeds the objective value of solution y^{j*}, α_j^*, by assumption that $\alpha_i^* + g_{ij}\beta_i^* - \alpha_j^* > 0$, contradicting the optimality of y^{j*}.

For case (*c*), since there exists positive capacity on arc $(i,j) \in \mathcal{A}'$ and $(j,i) \in \mathcal{A}' \setminus \mathcal{A}$, we can employ our arguments for cases (*a*) and (*b*) to show that $\alpha_i^* + g_{ij}\beta_i^* = \alpha_j^*$. Thus, constraints (3.2c) are satisfied at equality for all positive-flow non-saturated arcs $(i,j) \in \mathcal{A}$.

Next, we define $\alpha_t^* = 1$ and show that computing α_s^* using model (3.6) satisfies constraint (3.2b). To do so, we need only to show that α_s^* must equal 0. By contradiction, when $\alpha_s^* = 1$ for any solution to (3.6), then there must exist an augmenting path from s to t in $\mathcal{A}'$ that does not visit an exhausted node. However, the APC would have discovered this path and augmented flow that saturated an arc or node capacity on that path by means of solving LPs (3.4) and (3.5). Thus, α_s^* must be strictly less than 1. Additionally, when $0 < \alpha_s^* < 1$, there must exist a path from s to t in $\mathcal{A}'$ visiting some set of exhausted nodes. To send flow through any of these exhausted nodes, there must exist a set of augmenting cycles increasing residual capacity at each exhausted node visited by this path. This contradicts our assumption that each s–t path p in $\mathcal{A}'$ visits some set of exhausted nodes in $\mathcal{V}_R$, and no capacity-increasing cycles exist that jointly increase the residual capacity of all exhausted nodes visited by p. Thus, α_s^* must equal 0, and (3.2b) is satisfied.

Finally, our definition of the duals satisfies constraints (3.2d) and (3.2e) for all $i \in \mathcal{V}$. The APC algorithm thus produces flows $\bar{x}$, for which there exists a complementary slack dual-feasible solution, when each s–t path p in $\mathcal{A}'$ visits some set of exhausted nodes in $\mathcal{V}$ and no capacity-increasing cycles exist that increase the residual capacity of each exhausted node visited by p. This proves that $\bar{x}$ is optimal. □

3.3 Heuristic APC Algorithm

Identifying a joint-cycle set discussed in Section 3.2.3 requires solving an LP since a combination of multiple cycles may be needed to increase the residual capacity of some exhausted nodes. Determining a flow adjustments for this set of cycles also necessitates the need for linear programming. Section 3.2.3 alternatively shows that determining flow adjustments is straightforward when only a single capacity-increasing cycle is needed to increase the residual capacity of a set of exhausted nodes. Thus, when solving a linear programming subroutine is not possible, we present the following the heuristic APC (h-APC) algorithm for identifying and augmenting flows on a set of capacity-increasing cycles that allow additional flow from s to t.

The h-APC is thus described as follows. After executing the m-AF algorithm to obtain feasible flows $\bar{x}$, the h-APC first checks the optimality of $\bar{x}$ by executing the dual-based algorithm found in Section 3.2.2. When flows $\bar{x}$ are not deemed optimal, the h-APC employs two subroutines for identifying capacity-increasing cycles and their respective flow adjustments: the cycle generation procedure (CGP) in Section 3.3.1 for identifying a set of capacity-increasing cycles Θ and the augmenting-flow subroutine (AFS) in Section 3.3.2 for determining the flow adjustment on each cycle in Θ that allows for the maximum amount of flow to be augmented on an s–t path.

To employ these subroutines, we enforce the condition that the set of capacity-increasing cycles Θ found during the CGP is *non-overlapping*. A set of cycles Θ is deemed non-overlapping when *(a)* all cycles in Θ are arc-disjoint; *(b)* for all nodes $\hat{i}$ visited by any cycle in Θ, only a single forward arc $(\hat{i}, j) \in \mathcal{A}' \cap \mathcal{A}$ or reverse arc $(k, \hat{i}) \in \mathcal{A}' \setminus \mathcal{A}$ is contained within any cycle in Θ; and *(c)* for all node pairs $j,\ k$ visited by any set of cycles in Θ, only one of the two such arcs $(j, k) \in \mathcal{A}' \cap \mathcal{A}$ or $(k, j) \in \mathcal{A}' \setminus \mathcal{A}$ may be contained within any cycle in Θ. This definition of Θ ensures that a capacity-increasing cycle in Θ increases the residual capacity of a set of exhausted nodes without decreasing the residual capacity of any nodes in Θ.

3.3.1 Cycle Generation Procedure

For the CGP, define parameters $g^u_{uv},\ g^v_{uv} \geq 0,\ \forall (u, v) \in \mathcal{A}$. If $(u, v) \in \mathcal{A}' \cap \mathcal{A}$ (i.e., (u, v) is a forward arc), then $g^u_{uv} = g_{uv}$ and $g^v_{uv} = 0$. Otherwise, if $(u, v) \in \mathcal{A}' \setminus \mathcal{A}$ (i.e., (u, v) is a reverse arc), then $g^u_{uv} = 0$ and $g^v_{uv} = g_{vu}$. Let the residual capacity of each arc $(u, v) \in \mathcal{A} \cap \mathcal{A}'$ be $\bar{c}_{uv} = c_{uv} - \bar{x}_{uv}$, and let the residual capacity of each arc $(u, v) \in \mathcal{A}' \setminus \mathcal{A}$ be $\bar{c}_{uv} = \bar{x}_{vu}$. Define $\overline{\mathcal{V}}_R$ to be the set of all

nodes $i \in \mathcal{V}_R$ for which the CGP has found a cycle that increases the residual capacity at node i, and let set $\mathcal{V}_\Theta \subseteq \mathcal{V}$ consists of all nodes visited by at least one cycle in Θ.

To identify a cycle that increases residual capacity at node $i \in \mathcal{V}_R$ without overlapping an existing cycle in Θ, define a graph $\mathcal{G}_i$ having a node set $\mathcal{V}_i$ and an arc set $\mathcal{A}_i$. Define $\mathcal{A}_\Theta \subseteq \mathcal{A}'$ as the set of all arcs contained within any cycle in Θ as well as all forward arcs $(\hat{u}, v) \in \mathcal{A}' \cap \mathcal{A}$ and reverse arcs $(w, \hat{u}) \in \mathcal{A}' \setminus \mathcal{A}$ for which $\hat{u} \in \mathcal{V}_\Theta$. Let $\mathcal{V}_i$ contain nodes $\overline{uv}$ for all arcs $(u, v) \in \{\mathcal{A}' \setminus \mathcal{A}_\Theta\} : \bar{c}_{uv} > 0$. For all arc pairs $(j, i), (i, k) \in \mathcal{A}'$ for which $\overline{ji},\ \overline{ik} \in \mathcal{V}_i$, let arc $(\overline{ji}, \overline{ik})$ belong to $\mathcal{A}_i$ when $g^i_{ji} - g^i_{ik} > 0$. The strict inequality in this definition reflects the requirement that residual capacity at node i must strictly increase. For all arc pairs $(u, v),\ (v, w) \in \mathcal{A}' : v \in \mathcal{V}_R \setminus \{i\}$, let arc $(\overline{uv}, \overline{vw})$ belong to $\mathcal{A}_i$ when $g^v_{uv} - g^v_{vw} \geq 0$ and $\overline{uv},\ \overline{vw} \in \mathcal{V}_i$. Finally, for all arc pairs $(u, v),\ (v, w) \in \mathcal{A}' : v \in \mathcal{V} \setminus \mathcal{V}_R$, let arc $(\overline{uv}, \overline{vw})$ belong to $\mathcal{A}_i$ when $\overline{uv},\ \overline{vw} \in \mathcal{V}_i$.

Any cycle over arcs in $\mathcal{A}'$ containing a reverse arc $(j, i) \in \mathcal{A}' \setminus \mathcal{A}$ corresponds to a cycle over arcs in $\mathcal{A}_i$ visiting node $\overline{ji} \in \mathcal{V}_i$. Furthermore, any cycle visiting $\overline{ji} \in \mathcal{V}_i$ must also visit some node $\overline{ik} \in \mathcal{V}_i$ for which $g^i_{ji} > g^i_{ik}$ because arc $(\overline{ji}, \overline{ik}) \in \mathcal{A}_i$. Such a cycle is therefore capacity-increasing at node i. Our definition of $\mathcal{V}_i$ ensures that the set of cycles Θ found by the CGP is non-overlapping.

The CGP proceeds as follows. The CGP first constructs $\mathcal{G}_i$ for a node $i \in \mathcal{V}_R \setminus \mathcal{V}'_R$, in which $\mathcal{V}'_R$ consists all nodes $i \in \mathcal{V}_R$ for which the CGP has already searched for a capacity-increasing cycle at node i. The CGP then searches for a cycle over arcs in $\mathcal{A}_i$ originating at some node $\overline{ji} \in \mathcal{V}_i$ corresponding to a reverse arc $(j, i) \in \mathcal{A}' \setminus \mathcal{A}$. In the algorithm that formally defines this process below, set ϕ contains all nodes $\overline{ji} \in \mathcal{V}_i : (j, i) \in \mathcal{A}' \setminus \mathcal{A}$ for which the CGP has already searched for a capacity-increasing cycle originating at node $\overline{ji}$. Let set θ_k consist of all arcs contained within the k^{th} cycle in Θ.

CGP Initialization: Set $k = 1,\ \theta_1 = \emptyset,\ \Theta = \emptyset,\ \mathcal{V}'_R = \emptyset$, and $\overline{\mathcal{V}}_R = \emptyset$, and proceed to CGP Step 1.

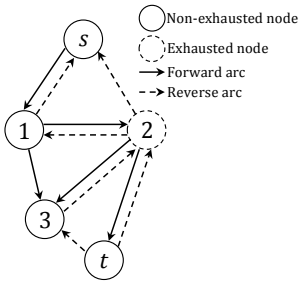
CGP Step 1: If $\mathcal{V}'_R = \mathcal{V}_R$, then proceed to CGP Step 3. Otherwise, for some node $i \in \mathcal{V}_R \setminus \mathcal{V}'_R$, construct $\mathcal{G}_i,\ \mathcal{V}_i$, and $\mathcal{A}_i$. Set $\phi = \emptyset$, place i in $\mathcal{V}'_R$, and proceed to CGP Step 2.

CGP Step 2: Identify a node $\overline{ji} \in \mathcal{V}_i \setminus \phi : (j, i) \in \mathcal{A}' \setminus \mathcal{A}$. If no such node exists, then return to CGP Step 1. Otherwise, place $\overline{ji}$ in ϕ and execute a DFS originating at $\overline{ji}$ over arcs in $\mathcal{A}_i$. If this search does not yield a capacity-increasing cycle, then repeat CGP Step 2. Otherwise, place all arcs

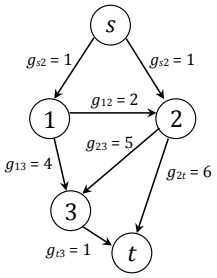
contained within the identified capacity-increasing cycle in sets θ_k and $\mathcal{A}_\Theta$ and place all nodes visited by this cycle in $\mathcal{V}_\Theta$. For all nodes $\hat{i} \in \mathcal{V}_\Theta$ visited by this cycle, place all forward arcs $(\hat{i}, j) \in \mathcal{A}' \cap \mathcal{A}$ and reverse arcs $(k, \hat{i}) \in \mathcal{A}' \setminus \mathcal{A}$ in $\mathcal{A}_\Theta$. Place all nodes $i \in \mathcal{V}_R \setminus \overline{\mathcal{V}}_R$ visited by this cycle in $\overline{\mathcal{V}}_R$, place all nodes $i \in \mathcal{V}_R \setminus \mathcal{V}'_R$ visited by this cycle in $\mathcal{V}'_R$, set $k = k + 1$, and return to CGP Step 1.

CGP Step 3: Execute a DFS to search for an s–t path $\bar{p}$ visiting only those nodes in $\{\mathcal{V} \setminus \mathcal{V}_R\} \cup \{\overline{\mathcal{V}}_R\}$. If no such path exists, we resort to executing the optimal augmenting-flow algorithm detailed in Section 3.2.3 to determine whether there exists some joint-cycle set. Otherwise, proceed to the AFS to determine the flow adjustment on each cycle in Θ and the flow on path $\bar{p}$.

We demonstrate the CGP on the residual network of arcs $\mathcal{A}'$ for flows $\bar{x}$ in Figure 3.6a, in which $\mathcal{V}_R$ contains only node 2.



(a) Residual network corresponding to flows $\bar{x}$



(b) Network displaying g-values

Figure 3.6: CGP implementation example

CGP Initialization: Set $k = 1,\ \theta_1 = \emptyset,\ \Theta = \emptyset,\ \mathcal{V}'_R = \emptyset,\ \overline{\mathcal{V}}_R = \emptyset$, and proceed to CGP Step 1.

CGP Step 1: Choose $i = 2 \in \mathcal{V}_R \setminus \mathcal{V}'_R$, and construct $\mathcal{G}_2,\ \mathcal{V}_2$, and $\mathcal{A}_2$. Figure 3.7 displays the transformed graph $\mathcal{G}_2$ of nodes $\mathcal{V}_2$ and arcs $\mathcal{A}_2$. Set $\phi = \emptyset$, place 2 in $\mathcal{V}'_R$, and proceed to CGP Step 2.

CGP Step 2: Place node $\overline{t2} \in \mathcal{V}_i \setminus \phi$ in ϕ since $(t, 2) \in \mathcal{A}' \setminus \mathcal{A}$. Execute a DFS originating at $\overline{t2}$ over arcs in $\mathcal{A}_i$. This search does not yield a capacity-increasing cycle, so repeat CGP Step 2.

CGP Step 2: Place node $\overline{32} \in \mathcal{V}_i \setminus \phi$ in ϕ since $(3, 2) \in \mathcal{A}' \setminus \mathcal{A}$. Execute a DFS originating at $\overline{32}$ over arcs in $\mathcal{A}_i$. This search yields the capacity-increasing cycle $(\overline{32}, \overline{21}, \overline{13})$. Place arcs $(3, 2),\ (2, 1)$, and $(1, 3)$ in sets θ_1 and $\mathcal{A}_\Theta$, and place nodes 1, 2, and 3 in $\mathcal{V}_\Theta$. Place forward arcs $(1, 2),\ (2, 3),\ (2, t)$

and reverse arcs $(3,1),\ (3,2),\ (t,2),\ (t,3)$ in $\mathcal{A}_\Theta$, place 2 in $\overline{\mathcal{V}}_R$, set $k=2$, and return to CGP Step 1.

CGP Step 1: Since $\mathcal{V}'_R = \mathcal{V}_R$, proceed to CGP Step 3.

CGP Step 3: Execute a DFS visiting only those nodes in $\{\mathcal{V} \setminus \mathcal{V}_R\} \cup \{\overline{\mathcal{V}}_R\} = \{s,1,2,3,t\}$ to determine the path $\bar{p} = (1,2,t)$.

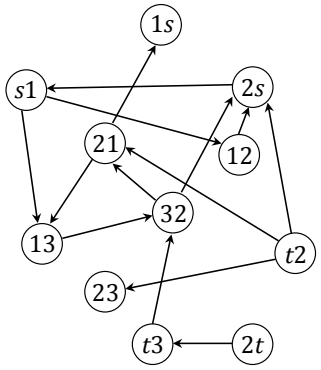


Figure 3.7: Transformed graph $\mathcal{G}_2$ for determining a capacity-increasing cycle at node 2.

3.3.2 Augmenting-flow Subroutine

Given the set of capacity-increasing cycles Θ and path $\bar{p}$ found at the conclusion of the CGP, the AFS seeks to determine a flow adjustment Δ_k on each cycle $\theta_k \in \Theta$ that allows for the maximum flow augmentation λ on $\bar{p}$. To do so, define $\mathcal{A}_{\bar{p}}$ as the set of all arcs contained within $\bar{p}$, $\mathcal{V}_{\bar{p}}$ as the set of all nodes visited by $\bar{p}$, and $\Theta_{\bar{p}} \subseteq \Theta$ as the set of all cycles visited by any node in $\mathcal{V}_{\bar{p}}$. Define $\mathcal{A}_{\Theta_{\bar{p}}}$ as the set of all arcs contained within any cycle in $\Theta_{\bar{p}}$, $\mathcal{V}_{\Theta_{\bar{p}}}$ as the set of all nodes visited by any cycle in $\Theta_{\bar{p}}$, $\mathcal{A}^k$ as the set of all arcs contained within a cycle $\theta_k \in \Theta_{\bar{p}}$, and $\mathcal{V}^k$ as the set of all nodes visited by a cycle $\theta_k \in \Theta_{\bar{p}}$. Define Θ_i (Θ_{uv}) as the set of cycles in $\Theta_{\bar{p}}$ that visit node $i \in \mathcal{V}_{\Theta_{\bar{p}}}$ (arc $(u,v) \in \mathcal{A}_{\Theta_{\bar{p}}}$). All Δ-values and λ must satisfy the following inequalities for the adjusted flows to remain feasible to (3.1):

$$\sum_{\theta_k \in \Theta_i} \left(\sum_{j:(i,j) \in \mathcal{A}^k \cap \{\mathcal{A}' \cap \mathcal{A}\}} g_{ij}\Delta_k - \sum_{h:(h,i) \in \mathcal{A}^k \cap \{\mathcal{A}' \setminus \mathcal{A}\}} g_{ih}\Delta_k \right) + \sum_{m:(i,m) \in \mathcal{A}_{\bar{p}} \cap \{\mathcal{A}' \cap \mathcal{A}\}} g_{im}\lambda - \sum_{\ell:(\ell,i) \in \mathcal{A}_{\bar{p}} \cap \{\mathcal{A}' \setminus \mathcal{A}\}} g_{i\ell}\lambda \leq \bar{b}_i,\ \forall i \in \mathcal{V}_{\bar{p}} \cup \mathcal{V}_{\Theta_{\bar{p}}}, \tag{3.8}$$

$$-\bar{c}_{vu} \leq \sum_{k \in \Theta_{uv}} \Delta_k - \sum_{m \in \Theta_{vu}} \Delta_m + \lambda \leq \bar{c}_{uv} \quad \forall (u,v) \in \mathcal{A}_{\bar{p}}, \tag{3.9}$$

$$-\bar{c}_{vu} \leq \sum_{k \in \Theta_{uv}} \Delta_k - \sum_{m \in \Theta_{vu}} \Delta_m \leq \bar{c}_{uv} \quad \forall (u,v) \in \mathcal{A}_{\Theta_{\bar{p}}} \setminus \mathcal{A}_{\bar{p}},\ (v,u) \notin \mathcal{A}_{\bar{p}}. \tag{3.10}$$

Inequalities (3.8) ensure that the capacity consumed at each node visited by $\bar{p}$ or any cycle in $\Theta_{\bar{p}}$ does not exceed its residual capacity. Inequalities (3.9) and (3.10) ensure that the net sum of flow adjustments does not exceed the residual arc capacity on each arc contained within $\bar{p}$ or any cycle in $\Theta_{\bar{p}}$. Inequalities (3.9) and (3.10) also ensure that the flow on each arc in $\mathcal{A}_{\bar{p}} \cup \mathcal{A}_{\Theta_{\bar{p}}}$ remains non-negative. The AFS thus determines values of $\Delta_k,\ \forall \theta_k \in \Theta_{\bar{p}}$, in order to maximize λ, subject to (3.8)–(3.10).

The AFS begins in Step 1 by establishing scalar upper bounds on λ. These bounds are derived from constraints (3.8) for nodes $i \in \mathcal{V}_{\bar{p}} \setminus \mathcal{V}_{\Theta_{\bar{p}}}$ and (3.9) for arcs $(u,v) \in \mathcal{A}_{\bar{p}} \setminus \mathcal{A}_{\Theta_{\bar{p}}}$ for which $(v,u) \notin \mathcal{A}_{\Theta}$. Step 2 then establishes scalar upper bounds on $\Delta_k,\ \forall \theta_k \in \Theta_{\bar{p}}$, based on (3.8) for nodes $i \in \mathcal{V}_{\Theta_{\bar{p}}} \setminus \mathcal{V}_{\bar{p}}$ and (3.10) for arcs $(u,v) \in \mathcal{A}_{\Theta_{\bar{p}}} \setminus \mathcal{A}_{\bar{p}}$ for which $(v,u) \notin \mathcal{A}_{\bar{p}}$. We refer to such constraints as *simple constraints*.

The AFS then analyzes constraints that involve both λ and Δ_k for some $\theta_k \in \Theta_{\bar{p}}$. These constraints are given by (3.8) for nodes $i \in \mathcal{V}_{\Theta_{\bar{p}}} \cap \mathcal{V}_{\bar{p}}$ and (3.9) for arcs $(u,v) \in \mathcal{A}_{\bar{p}} \cap \mathcal{A}_{\Theta_{\bar{p}}}$ and for $(u,v) \in \mathcal{A}_{\bar{p}} \setminus \mathcal{A}_{\Theta_{\bar{p}}},\ (v,u) \in \mathcal{A}_{\Theta_{\bar{p}}}$. We call these *linking constraints* in the following discussion. Since all cycles in $\Theta_{\bar{p}}$ are non-overlapping, these constraints contain only λ and a single Δ_k, as opposed to the multiple Δ-variables that would appear in these constraints if the cycles overlapped. Letting $g_u^{\bar{p}}$ (g_u^k) be the capacity consumed at node $u \in \mathcal{V}^k \cap \mathcal{V}_{\bar{p}}$ as a result of augmenting a single unit of flow along $\bar{p}$ ($\theta_k \in \Theta_{\bar{p}}$), define

$$g_u^{\bar{p}} = \sum_{v:(u,v) \in \mathcal{A}_{\bar{p}} \cap \{\mathcal{A}' \cap \mathcal{A}\}} g_{uv} - \sum_{w:(w,u) \in \mathcal{A}_{\bar{p}} \cap \{\mathcal{A}' \setminus \mathcal{A}\}} g_{uw},\ \forall u \in \mathcal{V}^k \cap \mathcal{V}_{\bar{p}},$$

$$g_u^k = \sum_{v:(u,v) \in \mathcal{A}^k \cap \{\mathcal{A}' \cap \mathcal{A}\}} g_{uv} - \sum_{w:(w,u) \in \mathcal{A}^k \cap \{\mathcal{A}' \setminus \mathcal{A}\}} g_{uw},\ \forall u \in \mathcal{V}^k \cap \mathcal{V}_{\bar{p}},\ \theta_k \in \Theta_{\bar{p}}.$$

The overlapping constraints can be written as follows for each cycle $\theta_k \in \Theta_{\bar{p}}$:

$$\lambda \leq \frac{\bar{b}_u - g_u^k \Delta_k}{g_u^{\bar{p}}},\ \forall u \in \mathcal{V}^k \cap \mathcal{V}_{\bar{p}},\ g_u^{\bar{p}} > 0, \tag{3.11a}$$

$$\lambda \geq \frac{g_u^k \Delta_k - \bar{b}_u}{-g_u^{\bar{p}}},\ \forall u \in \mathcal{V}^k \cap \mathcal{V}_{\bar{p}},\ g_u^{\bar{p}} < 0,\ g_u^k > 0, \tag{3.11b}$$

$$\lambda \leq \bar{c}_{uv} - \Delta_k,\ \forall (u,v) \in \mathcal{A}_{\bar{p}} \cap \mathcal{A}^k, \tag{3.11c}$$

$$\lambda \leq \bar{c}_{uv} + \Delta_k, \ \forall (u,v) \in \mathcal{A}_{\bar{p}} \setminus \mathcal{A}^k, \ (v,u) \in \mathcal{A}^k, \tag{3.11d}$$

$$\lambda \geq 0. \tag{3.11e}$$

If $g_u^{\bar{p}} = 0$ or both $g_u^{\bar{p}} < 0$ and $g_u^k \leq 0$, then constraints (3.8) do not provide a finite upper bound or a positive lower bound on λ. In either of these cases, constraints (3.8) are satisfied for any $\lambda \geq 0$ and $\Delta_k \geq 0$.

We now consider the problem, which we term SUB_k, of maximizing λ subject to constraints (3.11a)–(3.11e), given $\theta_k \in \Theta_{\bar{p}}$. Constraints (3.11a), (3.11c), and (3.11d) imply that λ is bounded above by a piecewise linear concave function of Δ_k, and constraints (3.11b) and (3.11e) imply that λ is bounded below by a piecewise linear convex function of Δ_k. Define λ'_k and Δ'_k as the values of λ and Δ_k, respectively, in an optimal solution to SUB_k. For each cycle $\theta_k \in \Theta_{\bar{p}}$, the AFS optimizes SUB_k in Step 3 and returns values λ'_k and Δ'_k or reports that the problem is unbounded. Each optimal SUB_k solution bounds the value of λ (in addition to the bounds on λ established in Step 1), and Δ_k is bounded above by the minimum of Δ'_k and the upper bound on Δ_k established in AFS Step 2. AFS Step 4 uses these observations to establish optimal values for $\Delta_k, \ \forall \theta_k \in \Theta_{\bar{p}}$, and λ, before updating $\bar{x}_{ij}, \ \forall (i,j) \in \mathcal{A}'$. For the organization of AFS Step 3, let

- $\tilde{\Theta}$ be the set of all cycles $\theta_k \in \Theta_{\bar{p}}$ for which bounds on λ and Δ_k have been established based on linking constraints,
- $\mathcal{C}_k^{UB}$ be the set of all constraints (3.11a), (3.11c), or (3.11d) that form the upper concave hull of SUB_k,
- $\mathcal{C}_k^{LB}$ be the set of all constraints (3.11b) or (3.11e) that form the lower convex hull of SUB_k,
- e_k represent the best current solution to SUB_k,
- the *slope* of each constraint in (3.11a) or (3.11b) be defined as $-g_u^k / g_u^{\bar{p}}$
- the *slope* of each constraint in (3.11c) and each constraint in (3.11d) be -1 and 1, respectively,
- $\bar{c}$ be the constraint formed by the bound Δ_k^{ub} found in AFS Step 2 having a *slope* defined as ∞,
- λ_{ub} and Δ_k^{ub} represent the upper bounds on λ and $\Delta_k, \ \forall \theta_k \in \Theta_{\bar{p}}$, respectively, based on (3.8)–(3.10).

AFS Step 1: Compute λ_{ub} as the minimum value of $\bar{c}_{uv}$ over all arcs $(u,v) \in \mathcal{A}_{\bar{p}}$ corresponding to simple constraints (3.9). If there are no such constraints, then set $\lambda_{ub} = \infty$. Next, set $\lambda_{ub} = \min\{\lambda_{ub},\ \min_{i \in \{\mathcal{V}_{\bar{p}} \setminus \mathcal{V}_{\Theta_{\bar{p}}}\}}\{\bar{b}_i / g_i^{\bar{p}}\}\}$, where the inner minimization evaluates to ∞ if $\mathcal{V}_{\bar{p}} \setminus \mathcal{V}_{\Theta_{\bar{p}}} = \emptyset$. Proceed to AFS Step 2.

AFS Step 2: For each $\theta_k \in \Theta_{\bar{p}}$, compute Δ_k^{ub} as the minimum value of $\bar{c}_{uv}$ over all arcs $(u,v) \in \mathcal{A}^k$ corresponding to simple constraints (3.10). If there are no such constraints, then set $\Delta_k^{ub} = \infty$. Next, set $\Delta_k^{ub} = \min\{\Delta_k^{ub},\ \min_{i \in \mathcal{V}^k \setminus \mathcal{V}_{\bar{p}}}\{\bar{b}_i / g_i^k\}\}$ for each $\theta_k \in \Theta_{\bar{p}}$. (The inner minimization also evaluates to ∞ if $\mathcal{V}^k \setminus \mathcal{V}_{\bar{p}} = \emptyset$.) Set $\tilde{\Theta} = \emptyset$, and proceed to AFS Step 3.

AFS Step 3: If $\tilde{\Theta} = \Theta_{\bar{p}}$, then proceed to AFS Step 3d. Otherwise, identify a cycle $\theta_k \in \Theta_{\bar{p}} \setminus \tilde{\Theta}$, and proceed to AFS Step 3a.

AFS Step 3a: Determine the constraints $\mathcal{C}_k^{UB}$ that form the concave upper hull for SUB_k, and sort the constraints $\mathcal{C}_k^{UB}$ in decreasing order according to slope. Determine the constraints $\mathcal{C}_k^{LB}$ that form the convex lower hull for SUB_k. Proceed to AFS Step 3b. (Appendix A provides an algorithm for determining sets $\mathcal{C}_k^{UB}$ and $\mathcal{C}_k^{LB}$.)

AFS Step 3b: If there exists a constraint in $\mathcal{C}_k^{UB}$ having positive slope, then let $\hat{c}$ be the constraint in $\mathcal{C}_k^{UB}$ having the minimum positive slope, and let ω be the slope of $\hat{c}$. Otherwise, let $\omega = 0$. Define $\hat{\mathcal{C}}$ to be the subset of constraints in $\mathcal{C}_k^{LB} \cup \{\bar{c}\}$ having a slope greater than ω, and sort $\hat{\mathcal{C}}$ in increasing order according to slope. If all constraints in $\mathcal{C}_k^{UB}$ have positive slope and $\hat{\mathcal{C}} = \emptyset$, then SUB_k is unbounded. In that case, set $\lambda_k' = \infty$ and $\Delta_k' = \infty$, and proceed to AFS Step 3d. Otherwise, set $\tilde{c}$ to be the constraint in $\mathcal{C}_k^{UB}$ having the maximum negative slope (i.e., the smallest absolute value of all negative slopes). Set e_k to be the intersection point of constraints $\hat{c}$ and $\tilde{c}$, and set $i = 0$. Proceed to AFS Step 3c.

AFS Step 3c: If $\hat{\mathcal{C}} = \emptyset$, then determine λ_k' and Δ_k' for solution e_k and proceed to AFS Step 3d. Otherwise, identify a constraint c in $\hat{\mathcal{C}}$ having minimum slope. If e_k is feasible to c, then remove c from $\hat{\mathcal{C}}$, set e_k equal to the intersection of constraints $\hat{c}$ and $\tilde{c}$, set $i = \max\{i-1, 0\}$, and repeat AFS Step 3c. Otherwise, if e_k is not feasible to c, then set $\tilde{c} = c$, $i = i+1$, $\hat{c}$ to be the constraint in $\mathcal{C}_k^{UB}$ having the i^{th} minimum positive slope, and e_k as the intersection of $\hat{c}$ and the constraint in $\mathcal{C}_k^{UB}$

having $(i+1)^{st}$ minimum positive slope, and repeat AFS Step 3c.

AFS Step 3d: If $\lambda'_k < \lambda_{ub}$, then set $\lambda_{ub} = \lambda'_k$. Place θ_k in $\tilde{\Theta}$, and return to AFS Step 3.

AFS Step 4: Set $\lambda = \lambda_{ub}$ and $\Delta_k = \Delta_k^{ub}, \forall \theta_k \in \Theta_{\bar{p}}$. Update flows $\bar{x}$ by setting

$$\bar{x}_{uv} = \bar{x}_{uv} + \lambda + \sum_{\theta_k \in \Theta_{uv}} \Delta_k - \sum_{\theta_k \in \Theta_{vu}} \Delta_k, \ \forall (u,v) \in \mathcal{A}_{\bar{p}} \cap \{\mathcal{A}' \cap \mathcal{A}\},$$

$$\bar{x}_{uv} = \bar{x}_{uv} - \lambda + \sum_{\theta_k \in \Theta_{uv}} \Delta_k - \sum_{\theta_k \in \Theta_{vu}} \Delta_k, \ \forall (v,u) \in \mathcal{A}_{\bar{p}} \cap \{\mathcal{A}' \setminus \mathcal{A}\}, \text{and}$$

$$\bar{x}_{uv} = \bar{x}_{uv} + \sum_{\theta_k \in \Theta_{uv}} \Delta_k - \sum_{\theta_k \in \Theta_{vu}} \Delta_k, \ \forall (u,v) \in \{\mathcal{A}_{\Theta_{\bar{p}}} \setminus \mathcal{A}_{\bar{p}}\} \cap \{\mathcal{A}' \cap \mathcal{A}\}.$$

Return to the optimality checking phase of the h-APC to check if flows $\bar{x}$ can be deemed optimal.

3.4 Computational Results

In this section we present computational results for the APC and the h-APC. We first compare the APC with LP (3.1) for solving the NCMFP over large networks. We then describe the results for the h-APC for smaller networks more prevalent in WSN applications. We solve all mathematical programming models using CPLEX 12.8 via ILOG Concert Technology. We performed all experiments on a computer having a 2.9GHz Dual-core Intel i7 processor with 8GB RAM. We present all CPU times in seconds and impose a 3600 second time limit and a 7GB memory limit.

For all instances, we consider randomly-generated integer c- and g-values uniformly distributed between $[1, 10]$ and $[1, 6]$, respectively. Additionally, for each instance, we consider a uniform b-value for all nodes in $\mathcal{V}$. We consider sparse networks having an average node out-degree of $|\mathcal{V}|/2$. We first generate five random instances of the NCMFP for networks having 100, 200, 300, 400, 500, 600, 700, and 800 nodes, in which $b_i = 50, \ \forall i \in \mathcal{V}$. We also generate five random instances of the NCMFP over networks having 800 nodes, in which $b = 30,\ 35,\ 40,\ 45$, and 55. Finally, we generate five random instances of the NCMFP over networks having 1000 nodes, in which $b = 30,\ 35,\ 40,\ 45,\ 50$, and 75. In all cases, we deem flows produced by the APC as optimal when the optimality gap is less than or equal to 0.01%. Thus, we set optimality gap to 0% for any APC solution whose optimality gap is $< 0.01\%$.

Tables 3.1 displays the computational results for the APC and model (3.1) when $b = 50$, Table 3.2 displays the computational results of the APC and (3.1) for networks having 800 nodes, and

Table 3.3 displays the computational results of the APC for networks having 1000 nodes. Each table depicts the average CPU time and the average number of calls to LPs (3.4) and (3.5) for the APC. Additionally, Tables 3.1 and 3.2 also depict the average optimality gap for the APC at termination.

Table 3.1: Average NCMFP results for $b = 50$

Number of Nodes	CPU Time (sec.)		Average Number of APC LPs	Average Opt. Gap
	LP	APC		
100	0.95	0.36	3.8	0%
200	8.67	2.66	9.4	0.06%
300	9.31	5.50	5.6	0%
400	23.78	13.28	9.0	0.07%
500	55.15	39.17	11.6	0.23%
600	96.11	39.78	13.2	0%
700	174.52	42.83	8.8	0.08%
800	308.46	87.20	14.4	0.09%

Table 3.1 displays the computational results for networks when $b = 50$. Both solution techniques required less than one second on average to solve the NCMFP over networks having 100 nodes. For all sizes of networks, the APC required less time than the LP. The LP required less than one minute to solve all instances of networks having 500 nodes or fewer, and the APC required less than one minute, on average, for networks having 700 nodes or fewer. The savings in computational time for the APC increases as the size of the network grows. For example, the LP requires about twice as much computational time as the APC for networks having 300 nodes. For networks having 800 nodes, the LP takes almost four times longer to solve the NCMFP than the APC.

On average, the APC produced an NCMFP solution within 0.25% of optimality for all instances of $b = 50$. Furthermore, the APC produced a solution within 0.09% of optimality for all network sizes, except for those having 500 nodes. The larger optimality gap for networks having 500 nodes was due to one instance that terminated with a solution having an optimality gap around 1%. The APC was able to determine an optimal solution for all instances of networks having 100, 300, or 600 nodes. For all other sizes of networks, the positive average optimality gap is a result of one or two instances terminating with a small positive optimality gap.

Table 3.2 displays the computational results for networks having 800 nodes. As the value of b increases from 30 to 35, the computational time for the APC also increases since the APC solved models (3.4) and (3.5) more often before termination. The small number of calls to the LP for

Table 3.2: Average NCMFP results over networks having 800 nodes

b	CPU Time (sec.)		Average Number of APC LPs	Average Opt. Gap
	LP	APC		
30	296.0	45.1	4.4	0.92%
35	319.1	163.7	23.6	0.92%
40	293.8	124.5	20.0	0.35%
45	283.8	55.9	7.8	0.20%
50	308.5	87.2	14.4	0.09%
55	287.8	73.6	13	0%
75	265.3	12.4	0	0%

$b = 30$ is likely due to the fact that the APC terminated early with a sub-optimal solution. Thus, the likelihood that the APC determines an optimal NCMFP solution increases as the b-value increases.

Solving models (3.4) and (3.5) requires more time when more capacity-increasing cycles exist in the residual network that increase capacity at exhausted nodes. Thus, the computational time of the APC is more dependent on the number of times models (3.4) and (3.5) are solved since these models only require a few seconds to solve, on average. The computational time for the APC decreases as the number of times (3.4) and (3.5) are solved also decreases. For example, the APC required solving (3.4) and (3.5) most frequently when $b = 35$, which results in the highest average computational time among all values of b. When $b \geq 75$, the computational time for the APC remained around 12 seconds, and our dual heuristic determined a dual feasible solution corresponding to a minimum cut in all such instances. The APC may be especially useful when a large network contains a relatively small set of particular nodes that are susceptible to exhaustion. Alternatively, the computational time for the LP requires approximately 293 seconds for all values of b.

Table 3.3: Average APC results over networks having 1000 nodes

b	CPU Time (sec.)	Average Number of APC LPs
30	101.3	6.4
35	208.7	18.4
40	231.8	25
45	188.9	19
50	230.0	28.2
75	22.0	0

Table 3.3 displays the computational results for networks having 1000 nodes. A preliminary investigation revealed that model (3.1) was unable to find an optimal solution for most instances

before reaching memory limit for networks having 1000 nodes. The APC requires less computational time for networks having lower values of b. The average computational time for the APC increased as the value of b increased until reaching its maximum average computational time when $b = 40$. The computational time then decreased as b increased from 40 to 45. As with networks having 800 nodes, this trend in computational time is likely due to the fact that the APC terminates early with a sub-optimal solution for lower values of b. The computational time for instances when $b = 50$ was particularly large as a single instance required over 600 seconds to solve.

The instances requiring the most computational time also required the most number of calls to (3.4) and (3.5). Thus, as observed with networks having 800 nodes, the computational time for the APC increases as the number of times (3.4) and (3.5) are solved also increases. Finally, for $b = 75$, the APC only required roughly 22 seconds since our dual heuristic algorithm determined a feasible set of duals without solving LPs (3.4) or (3.5). Overall, these results show that the APC appears to be an attractive alternative to (3.1) for determining a quality solution to the NCMFP over large networks having 800 nodes or more. For many of these instances, the APC was able to determine an optimal NCMFP solution. In all other instances, the APC was able to determine a solution within 1% of optimality, on average, while requiring significantly less computational time and memory.

To explore the effectiveness of the h-APC, we next generate five random instances of the NCMFP over networks having 100 nodes for $b = 40,\ 45,\ 50,\ 55$, and 60. Table 3.4 depicts the average CPU time, the average number of calls to the AFP subroutine, and the average optimality gap at termination of the h-APC. The optimality gap in this case refers to the relative percent difference between the flow value determined by the h-APC and the optimal maximum flow value.

Table 3.4: Average h-APC results over networks having 100 nodes

b	CPU Time (sec.)	Average Number of Calls to AFP	Average Opt. Gap
40	136.53	10.4	0.41%
45	205.12	18.8	0.39%
50	132.76	7.4	0.13%
55	94.05	8.2	0.05%
60	0.01	0	0%

Table 3.4 displays the computational results for the h-APC over networks having 100 nodes. The h-APC required the most computational time when $b = 45$. The h-APC also required the most calls to the AFP subroutine for this value of b. As the b-value increased beyond 45, the average

computational time and the number of calls to the AFP decrease. Thus, results show that the h-APC requires less time and is more likely to determine an optimal solution for higher values of b and that the computational time likely depends on the number of calls to the AFP subroutine. When $b \geq 60$, the dual heuristic determined a feasible set of dual values corresponding to a minimum cut without executing the h-APC.

The h-APC determined a flow within a 1% of the maximum flow in all instances. Most instances were solved within two minutes having a flow within 0.5% of the maximum flow. As b increases, the average optimality gap alternatively decreases. For example, the h-APC produced a flow within 0.25% of the maximum flow in all instances for $b = 50$. Two out of five such instances were solved to optimality within 90 seconds. Alternatively, the LP and the APC required less than a second to solve the NCMFP over networks having 100 nodes. The CGP and AFP subroutines within the h-APC require too much computational time to be considered useful. Thus, the h-APC is not an attractive solution technique for the NCMFP in most cases. However, the h-APC may be a practical solution technique in the case that no linear programming solver is available.

Chapter 4

Models and Algorithms for Solving Maximum Flow Problems Having Semicontinuous Path-flow Restrictions in Simultaneous Flow Settings

4.1 Introduction and Problem Description

Consider a variation of the maximum flow problem (MFP) having node and arc capacities, along with semicontinuous flow restrictions. These problems take place on a graph $\mathcal{G}$ having nodes $\mathcal{V}$ and arcs $\mathcal{A}$. Contained within $\mathcal{V}$ is a source node s and a sink node t. For every node $i \in \mathcal{V} \setminus \{s,t\}$, there exists a directed path from s to i and from i to t. Let $N \in \mathbb{Z}^{|\mathcal{V}|\times|\mathcal{A}|}$ be the node-incidence matrix of $\mathcal{G}$, where each column corresponding to arc $(i,j) \in \mathcal{A}$ has exactly two non-zero entries: 1 in row i and -1 in row j. Let $n \in \mathbb{Z}^{|\mathcal{V}|}$ be the vector of supply/demand values, in which $n_s = 1,\ n_t = -1$, and $n_i = 0,\ \forall i \in \mathcal{V} \setminus \{s,t\}$.

Let $c_{ij} > 0$ be the capacity of each arc $(i,j) \in \mathcal{A}$. For this problem we also define $b_i > 0$ as the capacity for node $i \in \mathcal{V}$. Each unit of flow on arc $(i,j) \in \mathcal{A}$ consumes one unit of capacity on (i,j), and $g_{ij} \geq 0$ units of capacity at node i. Scalar variable z corresponds to the maximum flow through the network, and variables x_{ij} represent the amount of flow on arc $(i,j) \in \mathcal{A}$. The MFP without semicontinuous flow restrictions can be formulated by the following linear programming (LP)

model.

$$\max\ z \tag{4.1a}$$

$$\text{s.t. } Nx - nz = 0 \tag{4.1b}$$

$$\sum_{j:(i,j)\in\mathcal{A}} g_{ij}x_{ij} \leq b_i \quad \forall i \in \mathcal{V} \tag{4.1c}$$

$$x_{ij} \leq c_{ij} \quad \forall (i,j) \in \mathcal{A} \tag{4.1d}$$

$$x \geq 0 \tag{4.1e}$$

The objective function (4.1a) maximizes flow transmitted in the network, while constraints (4.1b) ensure flow balance. Constraints (4.1c) and (4.1d) correspond to node and arc capacity constraints, respectively. Constraints (4.1e) enforce flow non-negativity. Throughout, we assume without loss of generality that there exists an optimal solution to (4.1).

As described in Chapter 1, flows that satisfy semicontinuous restrictions with respect to a given set of variables are said to be *stable*. The simplest case enforces stability restrictions on arc flows, which can be achieved by defining a binary variable $y_{ij},\ \forall (i,j) \in \mathcal{A}$, and restricting $uy_{ij} \geq x_{ij} \geq \ell y_{ij}$ [Beale, 1979, 1980, 1985]. Instead, we examine problems having stability restrictions on network paths, in which a *path* refers to a common amount of flow from any source node to any demand node using a sequence of arcs that visits each node at most once. Assuming all flows are simultaneously transmitted, we require that flow f_p on each path p, if positive, must be greater than or equal to ℓ. Any solution satisfying these restrictions are deemed *static-stable*.

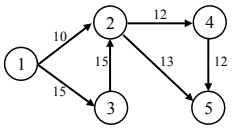
The rest of the chapter is organized as follows. Section 4.2 presents an MIP formulation that is pseudo-polynomial in size and a B&P algorithm for the MFP having static-stable restrictions. In Section 4.3 we present the computational results of our algorithms.

4.2 Problem Definition and Formulations

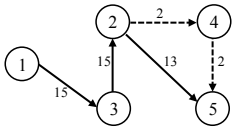
In Section 4.2.1 we examine the MFP with static-stable restrictions. We present two formulations for this problem, and show how to solve a relaxation of one of our formulations by column generation. In Section 4.2.2 we present a branching mechanism for our B&P algorithm.

4.2.1 Static Stability Formulations

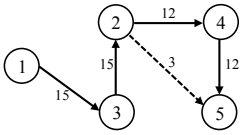
We begin by examining the maximum flow problem with static-stable restrictions (MFP-S). These restrictions imply arc stability, because if all positive path flows are at least as large as ℓ, then all arc flows must be as well. However, arc-stable restrictions do not necessarily imply static stability, as demonstrated by the network flow in Figure 4.1, where $\ell = 10$, node 1 is the lone source node, and node 5 is the lone sink node.



(a) Example flow with stable arc flows



(b) Remaining flows after p_1



(c) Remaining flows after p_2

Figure 4.1: Static stability example with $\ell = 10$

The network flow in Figure 4.1a satisfies arc-stable restrictions. Now, suppose that there exists a static-stable solution. Such a solution must include arc $(1, 2)$ on some path to node 5. There are two such paths: $p_1 = (1, 2, 4, 5)$ and $p_2 = (1, 2, 5)$. Figure 4.1b depicts the remaining flows after sending flow on p_1. A path having no more than 2 units of flow must include arcs $(2, 4)$ and $(4, 5)$, and such a path would not be stable. If arc $(1, 2)$ is included in path p_2 (Figure 4.1c), then the next path including arc $(2, 5)$ would not be stable. Therefore, no static-stable solution exists corresponding to the flows in Figure 4.1.

We formulate an MIP model for the MFP-S. Letting $\bar{z}$ be the maximum flow sent in an optimal solution to (4.1), which provides an upper bound on the optimal MFP-S objective since semicontinuity restrictions are relaxed, an upper bound on the number of paths that can be used in any MFP-S solution is given by $\lfloor \bar{z}/\ell \rfloor$. Thus, let $S = \{1, \ldots, \lfloor \bar{z}/\ell \rfloor\}$ be an index set of at most $\lfloor \bar{z}/\ell \rfloor$ s–t paths. Continuous variables ϕ_p and x_{ij}^p represent the amount of flow on path $p = 1, \ldots, \lfloor \bar{z}/\ell \rfloor$ and the amount of flow on arc $(i, j) \in \mathcal{A}$ in path p, respectively. Binary variable y_{ij}^p equals 1 if and only if arc (i, j) has positive flow in path p.

$$\max z \tag{4.2a}$$

$$\text{s.t.} \sum_{p\in S}\sum_{i:(s,i)\in\mathcal{A}} x^p_{si} - z = 0 \tag{4.2b}$$

$$-\sum_{p\in S}\sum_{j:(j,t)\in\mathcal{A}} x^p_{jt} + z = 0 \tag{4.2c}$$

$$\sum_{p\in S}\sum_{j:(i,j)\in\mathcal{A}} g_{ij}x^p_{ij} \le b_i \quad \forall i \in \mathcal{V} \tag{4.2d}$$

$$\sum_{p\in S} x^p_{ij} \le c_{ij} \quad \forall (i,j) \in \mathcal{A} \tag{4.2e}$$

$$\sum_{j:(i,j)\in\mathcal{A}} y^p_{ij} - \sum_{h:(h,i)\in\mathcal{A}} y^p_{hi} = 0 \quad \forall i \in \mathcal{V}\setminus\{s,t\},\ p \in S \tag{4.2f}$$

$$x^p_{ij} = \phi_p y^p_{ij} \quad \forall p \in S,\ (i,j) \in \mathcal{A} \tag{4.2g}$$

$$\sum_{i:(s,i)\in\mathcal{A}} y^1_{si} = 1 \tag{4.2h}$$

$$\sum_{i:(s,i)\in\mathcal{A}} y^p_{si} \le \sum_{i:(s,i)\in\mathcal{A}} y^{p-1}_{si} \quad \forall p = 2,\dots,\lfloor \bar{z}/\ell \rfloor \tag{4.2i}$$

$$\phi_p \le \phi_{p-1} \quad \forall p = 2,\dots,\lfloor \bar{z}/\ell \rfloor \tag{4.2j}$$

$$\sum_{i:(s,i)\in\mathcal{A}} \ell y^p_{si} \le \phi^p \le \sum_{i:(s,i)\in\mathcal{A}} c_{si} y^p_{si} \quad \forall p \in S \tag{4.2k}$$

$$\phi_p \ge 0 \quad \forall p \in S \tag{4.2l}$$

$$y^p_{ij} \in \{0,1\} \quad \forall p \in S,\ (i,j) \in \mathcal{A} \tag{4.2m}$$

The objective function (4.2a) and constraints (4.2b) and (4.2c) maximize flow from s to t. Constraints (4.2d) and (4.2e) enforce node and arc capacity constraints, respectively. Constraints (4.2f) ensure that there exists an equal number of positive inflow and outflow arcs at each node in $\mathcal{V}\setminus\{s,t\}$ on each path p, while nonlinear constraints (4.2g) ensure that the flow on each arc (i,j) in path p equals ϕ_p. Constraints (4.2f) and (4.2g) enforce flow balance, although they do not prevent subtours. However, since $g_{ij} \ge 0, \forall (i,j) \in \mathcal{A}$, these subtours can be removed from any solution to (4.2) *a posteriori* without affecting feasibility or the objective function value. Note that constraints (4.2g) can be linearized by replacing $\phi_p y^p_{ij}$ with an auxiliary variable $u^p_{ij},\ \forall p = 1,\dots,\lfloor \bar{z}/\ell \rfloor,\ (i,j) \in \mathcal{A}$, and adding the following McCormick linearization constraints [McCormick, 1983]:

$$u^p_{ij} \ge 0$$

$$u^p_{ij} \ge \phi^p_{ij} + M(y^p_{ij} - 1)$$

$$u_{ij}^p \leq \phi_{ij}^p$$
$$u_{ij}^p \leq M y_{ij}^p$$

for all $p = 1, \ldots, \lfloor \bar{z}/\ell \rfloor,\ (i,j) \in \mathcal{A}$. These constraints effectively enforce $u_{ij}^p = \phi_p$ when $y_{ij}^p = 1, \forall p = 1, \ldots, \lfloor \bar{z}/\ell \rfloor,\ (i,j) \in \mathcal{A}$, and 0 otherwise. We set M equal to the maximum arc capacity among all arcs $(i,j) \in \mathcal{A}$.

Constraints (4.2h)–(4.2j) are symmetry-breaking constraints, as we describe in more detail below. Constraints (4.2k) guarantee static stability, while constraints (4.2l) and (4.2m) enforce non-negativity and binary restrictions on variables ϕ_p and y_{ij}^p, respectively. Note that model (4.2) is pseudo-polynomial in size, having $\mathcal{O}(\lfloor \bar{z}/\ell \rfloor |\mathcal{A}|)$ integer and continuous variables, along with $\mathcal{O}(\lfloor \bar{z}/\ell \rfloor |\mathcal{A}|)$ structural constraints.

Without the presence of symmetry-breaking constraints, we could reshuffle the path indices for a given solution and obtain an equivalent symmetrical solution. Thus, the branch-and-bound (B&B) tree would be forced to explore many mirror-image solutions during the search process, which substantially increases computational effort. Symmetry-breaking constraints enforce hierarchies that eliminate some (or all) symmetrical solutions. (For a full description of symmetry-breaking constraints, see [Sherali and Smith, 2001].) In this formulation, constraints (4.2h)–(4.2j) ensure that the flow on path $p-1$ is at least as large as the flow on path p for all $p = 2, \ldots, \lfloor \bar{z}/\ell \rfloor$.

Model (4.2) can be solved using B&B. However, the linear programming relaxation of (4.2) is weak, as the y-variables can fractionate in order to accommodate paths that do not satisfy the semicontinuous restrictions. Also, the MIP likely contains a relatively small number of positive-flow paths relative to $\lfloor \bar{z}/\ell \rfloor$. Thus, as an alternative, we propose a B&P algorithm that couples column generation (CG) with a branching mechanism to enforce static stability. For the CG scheme, we define P as the set of all possible s–t paths. After we relax stability restrictions and include only some subset of paths $\bar{P} \subset P$, we obtain the following continuous restricted master problem (RMP), in which the semicontinuity restrictions are omitted, effectively representing the linear programming relaxation of the problem. Let variables λ_p represent the flow on path $p \in \bar{P}$, and define z as before. Let parameter v_{ij}^p equal 1 if arc $(i,j) \in \mathcal{A}$ is in path $p \in \bar{P}$. The RMP is formulated as follows:

$$\max\ z \tag{4.3a}$$
$$\text{s.t.} \sum_{p \in \bar{P}} \sum_{i:(s,i) \in \mathcal{A}} v_{si}^p \lambda_p - z = 0 \qquad (\alpha_s) \tag{4.3b}$$

$$-\sum_{p\in\bar{P}}\sum_{j:(j,t)\in\mathcal{A}} v_{jt}^p\lambda_p + z = 0 \qquad (\alpha_t) \tag{4.3c}$$

$$\sum_{p\in\bar{P}}\sum_{j:(i,j)\in\mathcal{A}} g_{ij}v_{ij}^p\lambda_p \le b_i \quad \forall i\in\mathcal{V} \qquad (\beta_i) \tag{4.3d}$$

$$\sum_{p\in\bar{P}} v_{ij}^p\lambda_p \le c_{ij} \quad \forall (i,j)\in\mathcal{A} \qquad (\gamma_{ij}) \tag{4.3e}$$

$$\lambda_p \ge 0 \quad \forall p\in\bar{P}. \tag{4.3f}$$

The objective function (4.3a) and constraints (4.3b) and (4.3c) maximize flow in the network. Constraints (4.3d) and (4.3e) correspond to node and arc capacity constraints, respectively. Constraints (4.3f) enforce flow non-negativity. Note that because (4.3) contains no constraints involving semicontinuity restrictions, it is equivalent to formulation (4.1) when $\bar{P} = P$. We turn to the problem of branching to enforce semicontinuity in Section 4.2.2.

Let α_s, α_t, β_i, and γ_{ij} be the dual values for constraints (4.3b)–(4.3e), respectively. The LP subproblem (also called the *pricing problem*) seeks an s–t path p so that λ_p has a positive reduced cost in (4.3). Letting $\mathcal{A}_p \subseteq \mathcal{A}$ be the set of arcs having positive flow on path p, the negative of the reduced cost for p is

$$\alpha_s - \alpha_t + \sum_{(i,j)\in\mathcal{A}_p}(\beta_i g_{ij} + \gamma_{ij}). \tag{4.4}$$

We thus identify a path from s to t that minimizes (4.4). The pricing problem can then be modeled by the following shortest path problem, in which variables w_{ij} equal 1 if there exists flow on arc (i,j), and 0 otherwise.

$$\alpha_s - \alpha_t \; + \; \min \sum_{(i,j)\in\mathcal{A}} (\beta_i g_{ij} + \gamma_{ij})w_{ij} \tag{4.5a}$$

$$\text{s.t.} \sum_{i:(s,i)\in\mathcal{A}} w_{si} = 1 \tag{4.5b}$$

$$\sum_{j:(i,j)\in\mathcal{A}} w_{ij} - \sum_{h:(h,i)\in\mathcal{A}} w_{hi} = 0 \quad \forall i\in\mathcal{V}\setminus\{s,t\} \tag{4.5c}$$

$$\sum_{j:(j,t)\in\mathcal{A}} w_{jt} = 1 \tag{4.5d}$$

$$w_{ij} \ge 0 \quad \forall (i,j)\in\mathcal{A} \tag{4.5e}$$

Because the β- and γ-values are non-negative, no negative-cost arcs exist in the network. The pricing problem is thus solvable using Dijkstra's algorithm [Dijkstra, 1959].

Using the RMP and pricing problem, the CG algorithm is as follows. We first solve the RMP given some set of initial paths in $\bar{P}$, which can be arbitrarily chosen. We recover the dual values from the RMP to formulate the pricing problem objective (4.5a). If the optimal objective function value to the pricing problem is negative, then we have identified a path having a positive reduced cost. This path is added to $\bar{P}$ and the RMP is re-solved to generate a new set of dual values. This process repeats until no positive reduced-cost paths are identified, at which time we have recovered an optimal solution to the LP relaxation of the MFP-S. If any optimal flows are not stable, then we enforce a set of branching constraints to enforce stability, as described in Section 4.2.2.

4.2.2 Branching Mechanisms

When the optimal solution identified to the RMP formulation above is not static stable, the B&P algorithm enters a branching phase. Branching on λ-variables is ineffective: When some variable $\bar{\lambda}_p$ is less than ℓ, then setting $\bar{\lambda}_p = 0$ will generally result in the CG phase rediscovering path p. Adding constraints to prevent the regeneration of such columns destroys the totally unimodular constraint matrix structure in the pricing problem. To maintain efficient solution techniques, branching should be based on original arc-flow variables rather than variables in the RMP [Barnhart et al., 1995, Desrosiers et al., 1995]. Thus, our primary branching rule is based on aggregate arc flows over all paths in $\bar{P}$.

Branching is required for two particular cases with respect to aggregate arc flow. In Section 4.2.2.1 we present branching constraints when the aggregate flow on some arc (i, j) is not stable. We then present branching rules in Section 4.2.2.2 for the case in which the identified optimal RMP solution is not static stable, but the aggregate arc flows corresponding to this solution are arc stable.

4.2.2.1 Non-Stable Aggregate Arc Flows

Consider an arc (i, j) for which aggregate flow in the RMP solution is in the interval $(0, \ell)$. Our branching strategy attempts to enforce the condition that aggregate flow on (i, j) is either zero or no less than ℓ. Define $\mathcal{W}_{ij}$ to be the set of all paths in which arc $(i, j) \in \mathcal{A}$ has positive flow. On the down-branch, we ensure that no flow exists on (i, j) by removing all paths in $\mathcal{W}_{ij}$ from the RMP, and prohibiting the future generation of paths in $\mathcal{W}_{ij}$ by removing (i, j) from the pricing problem.

The up-branch would ideally require a flow of at least ℓ on (i, j); however, the dual values corresponding to these constraints would be non-positive, which makes the pricing problem NP-hard.

Instead, we employ an alternative strategy, which begins by determining a minimal cut-set $\mathcal{C}$ satisfying the conditions specified below.

Proposition 2. *Consider an RMP solution in which the aggregate-flow solution is acyclic, and in which the aggregate flow on arc $(i,j) \in \mathcal{A}$ is not stable. Define positive-flow arcs as those for which there is positive flow in the aggregate-flow solution. Let X contain node i and each node $m \in \mathcal{V}$ for which there exists a path using positive-flow arcs from m to i. Additionally, let $\bar{X} = \mathcal{V} \setminus X$. If set $\mathcal{C}$ contains all arcs (h,k) for which $h \in X$ and $k \in \bar{X}$, then $\mathcal{C}$ is a minimal cut-set containing (i,j). Furthermore, there exist no positive-flow arcs (u,v) in $\mathcal{C}$ for which $u \in \bar{X}$ and $v \in X$.*

Proof. Because there are no positive-flow cycles in the given RMP solution, node $j \in \bar{X}$ and $(i,j) \in \mathcal{C}$. Additionally, for every $m \in \mathcal{V}$ such that there exists a positive-flow path from m to i, there must also exist a positive-flow path from s to m. There also exists a path from each $k \in \bar{X}$ to t that does not use an arc in $\mathcal{C}$. To see this, recall that there exists a path from k to t by assumption. If that path used an arc $(u,v) \in \mathcal{C}$, then there is a path from k to u, along with a path from u to i (due to the fact that $u \in X$). But then, $k \in X$ as well, which is a contradiction. Therefore, all positive-flow paths from $k \in \bar{X}$ to t do not contain an arc in $\mathcal{C}$. Additionally, if any $(h,k) \in \mathcal{C}$ were omitted from $\mathcal{C}$, then there would exist a path from s to t containing arc (h,k).

Therefore, $\mathcal{C}$ is a minimal cut-set containing (i,j). Finally, if there were some arc (u,v) having positive flow such that $u \in \bar{X}$ and $v \in X$, then there must also exist a positive-flow path from u to i and u must be contained in X. Thus, there exist no positive-flow arcs (u,v) in $\mathcal{C}$ from $u \in \bar{X}$ to $v \in X$. This completes the proof. □

We determine a cut-set $\mathcal{C}$ by first removing any redundant cycles in the aggregate-flow solution, and then computing X as prescribed in Proposition 2. Letting $\hat{z}$ be the maximum flow value for an optimal RMP solution, note that the sum of aggregate flows $\sum_{(i,j)\in\mathcal{C}} v_{ij}^p \hat{\lambda}_p$ equals $\hat{z}$. We enforce the up-branch by adding the constraint:

$$\sum_{p\in\bar{P}} \sum_{(h,k)\in\mathcal{C}\setminus(i,j)} v_{hk}^p \lambda_p \leq \hat{z} - \ell. \tag{4.6}$$

Letting $\pi_{\mathcal{C}ij\hat{z}}$ be the (non-negative) dual associated with constraint (4.6) written for arc (i,j), cut $\mathcal{C}$, and bound $\hat{z}$, the objective function of the pricing problem now becomes:

$$\alpha_s - \alpha_t \; + \; \min \sum_{(i,j)\in\mathcal{A}} (\beta_i g_{ij} + \gamma_{ij}) w_{ij} + \sum_{(h,k)\in\mathcal{C}\setminus(i,j)} \pi_{\mathcal{C}ij\hat{z}} w_{hk}.$$

Since all π-values are non-negative, problem (4.5) can still be solved by Dijkstra's algorithm on the up-branch.

We now demonstrate that any static-stable solution having positive aggregate flow on arc (i,j) satisfies (4.6).

Proposition 3. *Define* MP′ *as formulation* (4.3) *with* $\bar{P} = P$*, augmented with inequality* (4.6) *corresponding to* $\mathcal{C}$*,* (i, j)*, and* $\hat{z}$*. Let* $(\bar{x}, \bar{z})$ *be an aggregate flow solution to problem* (4.1) *in which* $\bar{x}_{ij} \geq \ell$*. Then there exists a solution,* $\bar{\lambda}$*, to* MP′ *such that* $\sum_{p \in \bar{P}} v^p_{hk} \bar{\lambda}_p = \bar{x}_{hk},\ \forall (h, k) \in \mathcal{A}$*.*

Proof. Consider any feasible flow decomposition solution $\bar{\lambda}$ corresponding to $\bar{x}$ (known to exist due to the flow decomposition theorem [Ahuja et al., 1993]). Then, noting that $\mathcal{C}$ is a cut-set in $\mathcal{G}$ we have that $\sum_{p \in \bar{P}} \sum_{(h,k) \in \mathcal{C}} v^p_{hk} \bar{\lambda}_p \leq \hat{z}$, because $\hat{z}$ is an upper bound on the maximum flow. Since $\sum_{p \in \bar{P}} v^p_{ij} \bar{\lambda}_p \geq \ell$ due to the inclusion of (4.6) in *MP′*, we get that $\sum_{p \in \bar{P}} \sum_{(h,k) \in \mathcal{C} \setminus (i,j)} v^p_{hk} \bar{\lambda}_p$ must be no more than $\hat{z} - \ell$, as desired. This completes the proof. □

Corollary 1. *Consider a solution* $\bar{\lambda}$ *to* (4.3) *such that* $0 < \sum_{p \in \bar{P}} v^p_{ij} \bar{\lambda}_p < \ell$*. Constraint* (4.6) *cuts off* $\bar{\lambda}$*.*

Proof. The left-hand side of (4.6) evaluates to $\hat{z} - \sum_{p \in \bar{P}} v^p_{ij} \bar{\lambda}_p$, and so (4.6) is violated by $\ell - \sum_{p \in \bar{P}} v^p_{ij} \bar{\lambda}_p > 0$ units. □

Although constraint (4.6) written with respect to some (i, j), $\mathcal{C}$, and $\hat{z}$ cuts off the current non-stable RMP solution, it does not require the aggregate flow on (i, j) to be greater than or equal to ℓ. To see why, consider the example in Figure 4.2 with $\ell = 10$.

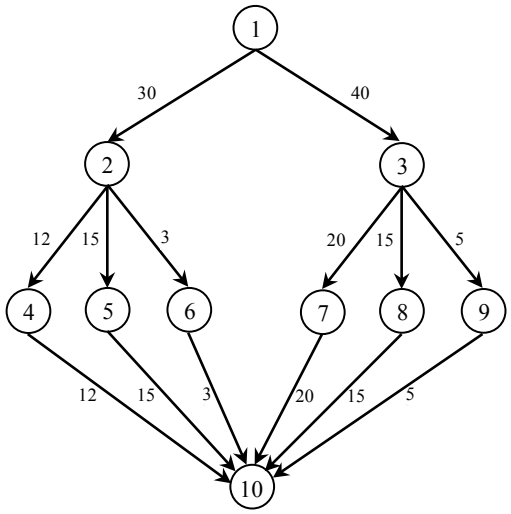


Figure 4.2: Non-stable aggregate arc-flow solution with $\ell = 10$

For the flows in Figure 4.2, the maximum flow value is $\hat{z} = 70$. If we branch on arc $(2, 6)$, then letting $X = \{1, 2\}$ and $\mathcal{C} = \{(2, 4), (2, 5), (2, 6), (1, 3)\}$, we add inequality $\sum_{p \in \bar{P}} \sum_{(h,k) \in \mathcal{C} \setminus (2,6)} v^p_{hk} \lambda_p \leq 60$ on the up-branch. Since the total flow among arcs $(h, k) \in \mathcal{C} \setminus (2, 6)$ is 67, inequality (4.6) cuts off the non-stable aggregate arc-flow solution in Figure 4.2. Note however that the next solution generated could be identical to the one in Figure 4.2, except with seven fewer units of flow on each of

the arcs $(1,3)$, $(3,7)$, and $(7,10)$. Thus, (4.6) cuts off the current RMP solution but does not force flow on (i,j) to be at least ℓ.

In the preceding example, after adding (4.6) corresponding to (i,j), $\mathcal{C}$, and $\hat{z}$ to RMP, we obtained a new solution for which (i,j) is still non-stable and $\mathcal{C}$ is still a cut-set. The objective, z', from this solution is strictly less than $\hat{z}$. If we branch again on (i,j), then there is no need to explore the down-branch (which was generated at the parent node of the B&P tree). The next up-branch could generate (4.6) corresponding to (i,j), $\mathcal{C}$, and z'. Because $z' < \hat{z}$, this new inequality implies the one generated for $\hat{z}$, and so (4.6) is merely updated by replacing $\hat{x}$ with z'.

In one special case, all arcs in $\mathcal{C}$ might emanate from a single node i. In that case, we can alternatively use the following constraint to enforce the up-branch condition:

$$\sum_{p\in\bar{P}} \sum_{(i,k)\in\mathcal{C}:k\neq j} g_{ik} v_{ik}^p \lambda_p \leq b_i - g_{ij}\ell. \tag{4.7}$$

This inequality is valid on the up-branch by the same logic as in Proposition 3. Inequality (4.7) may be particularly strong when g_{ij} is large relative to the other arcs $(i,k) \in \mathcal{C} : k \neq j$. We compare the strength of inequalities (4.6) and (4.7) using the following example. Letting $g_{12} = g_{13} = 1,\ g_{14} = 5,\ b_1 = 72$, and $\ell = 10$, consider the set of aggregate arc flows in Figure 4.3. For the flows in Figure

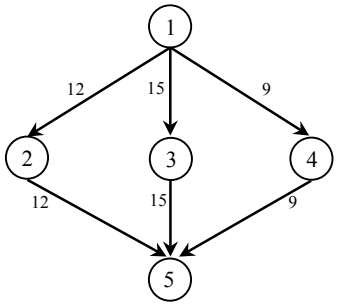


Figure 4.3: Non-stable aggregate arc-flow solution with $\ell = 10$

4.3, $\hat{z} = 36$ and arcs $(1,4)$ and $(4,5)$ are non-stable. Let $\mathcal{C} = \{(1,2),(1,3),(1,4)\}$ and examine non-stable arc $(1,4)$. Inequality (4.6) is given by $\sum_{p\in\bar{P}}(v_{12}^p + v_{13}^p)\lambda_p \leq 26$, and inequality (4.7) is given by $\sum_{p\in\bar{P}}(v_{12}^p + v_{13}^p)\lambda_p \leq 22$, which is tighter than (4.6).

Inequality (4.7) might not cut off the current optimal solution when $\sum_{p\in\bar{P}} \sum_{(i,k)\in\mathcal{C}:k\neq j} g_{ik} v_{ik}^p \bar{\lambda}_p + g_{ij}\ell < b_i$. Even when (4.7) cuts off the current optimal solution, inequalities (4.6) and (4.7) do not in general dominate one another. In our implementation, we investigate adding either (4.6), (4.7), or

both on an up-branch, depending on the extent to which each inequality is violated by the current optimal solution. In fact, it is possible that neither inequality is violated by a significant amount with respect to the current optimal RMP solution. In that case, we can directly branch up on a non-stable λ_p variable, as explained at the end of Section 4.2.2.2.

4.2.2.2 Stable Aggregate Arc Flows

Now consider the case in which aggregate arc flows in the RMP solution are stable. If all path flows in $\bar{P}$ are also stable, then the current solution to the RMP is feasible for the MFP-S, and no additional branching is required. Otherwise, there exists at least one path in $\bar{P}$ having a non-stable flow. In this case, there may or may not exist an alternative path decomposition corresponding to these aggregate flows that does indeed satisfy the static-stable restriction. The *stable path decomposition problem* (SPDP) is the problem of determining if a set of stable aggregate arc flows having non-stable path flows can be decomposed into a set of stable path flows.

Theorem 2. *The SPDP is NP-hard.*

The proof of Theorem 2 is provided in the Appendix B. Given Theorem 2, we now present a heuristic polynomial-time algorithm for solving SPDP. Given a set of stable aggregate arc flows $F_{ij},\ \forall (i,j) \in \mathcal{A}$, the algorithm is as follows. Define $r_{ij} \geq 0$ as the remaining flow to be sent on arc $(i,j) \in \mathcal{A}$, and initialize $r_{ij} = F_{ij},\ \forall (i,j) \in \mathcal{A}$. For the first step, find the minimum remaining flow $r_{min} = \min\{r_{ij} : (i,j) \in \mathcal{A}\}$. If $0 < r_{min} < \ell$, then the algorithm terminates and fails to provide an SPDP solution. Otherwise, find a path $\bar{p}$ from s to node t using only arcs (i,j) such that either $r_{ij} \geq \ell + r_{min}$ or $r_{ij} = r_{min}$; furthermore, at least one arc in the path should satisfy the latter condition. Since the graph of aggregate arc flows is directed and acylic, we can find a path satisfying the aforementioned conditions in polynomial time using depth-first search with a state variable recording whether or not an arc (i,j) has been traversed having $r_{ij} = r_{min}$. Again, if no such path exists, then the algorithm fails to identify an SPDP solution. Otherwise, reduce the flow r_{ij} on each arc (i,j) in $\bar{p}$ by r_{min} and return to the first step.

When the heuristic fails to decompose stable aggregate arc flows into a set of static-stable path flows, the stable aggregate arc-flow solution may still correspond to an SPDP solution. Rather than executing an exponential-time algorithm to determine if the SPDP has a solution, we instead simply branch directly on the λ-variables. For the up-branch, we enforce $\lambda_p \geq \ell$ for some non-stable path p, and we enforce $\lambda_p = 0$ on the down-branch. To avoid re-generating path p along the

down-branch, we add constraint

$$\sum_{(i,j)\in\mathcal{A}_p} w_{ij} \leq |\mathcal{A}_p| - 1, \tag{4.8}$$

to the pricing problem. With the addition of any constraints of the form (4.8), the pricing problem constraint matrix is no longer totally unimodular. Thus, we must now solve the IP formulation of pricing problem (4.5) to generate new paths. Hence, we replace (4.5e) with binary restrictions on the w-variables whenever (4.8) is added to the pricing problem.

4.3 Computational Results

In this section we present computational results for the MFP-S. We evaluate the performance and scalability of our B&P approach for the MFP-S in Sections 4.3.1 and 4.3.2, respectively. We implement our B&P algorithm in C using SCIP Optimization Suite 3.2.1 [Gerald et al., 2016]. We solve all mathematical optimization problems within the B&P using the SCIP default solver, SoPlex. All MIP instances are solved using CPLEX 12.6.2 via ILOG Concert Technology. We performed all experiments on a computer having a 2.9GHz Dual-core Intel i7 processor with 8GB RAM. We present all CPU times in seconds and impose a 3600 second time limit and 7GB memory limit.

4.3.1 Solving the MFP-S

We first analyze the performance of our B&P approach on the MFP-S using randomly-generated g-, b-, and c-values for dense networks having an arc between every distinct node pair except between s and t. Defining *relay* nodes to be those found in the set $\mathcal{V} \setminus \{s, t\}$, we generate five random instances of the MFP-S for each combination of networks having 10, 15, and 20 relay nodes and $\ell \in \{1, 3, 5\}$. For each network size and value of ℓ, the b-values, g-values, and c-values are random integers uniformly distributed between $[25, 100]$, $[1, 12]$, and $[10, 20]$, respectively.

We compare our B&P approach with model (4.2) when solving the MFP-S. Tables 4.1, 4.2, and 4.3 display the computational results of both approaches for networks having 10, 15, and 20 relay nodes, respectively. Each table depicts the average CPU time, average number of branches, the total number of instances solved given an optimality gap of 0.1%, the average optimality gap at termination, and the number of generated columns for the B&P. For those instances that are terminated due to the time limit, we record the number of branches in the B&B tree and the current optimality gap observed after 3600 seconds, and count the CPU time required for the instance as

3600 seconds.

Table 4.1 shows that, on average, the B&P algorithm requires no more than one second to solve the MFP-S for networks having 10 relay nodes. For all values of ℓ, the maximum CPU time using B&P is 2.3 seconds. While the average CPU time slightly increases from $\ell = 1$ to $\ell = 3$, we observe no significant change when increasing ℓ from 3 to 5. We note that our branching technique found only one instance in which an arc-stable solution did not correspond to a static-stable solution. For this instance, our SPDP heuristic algorithm (found in Section 4.2.2.2) was able to decompose this arc-stable solution into a static-stable solution.

Alternatively, Table 4.1 shows that, on average, model (4.2) requires almost 15 minutes to solve the MFP-S when $\ell = 1$, almost 30 minutes when $\ell = 3$, and about 41 minutes when $\ell = 5$. We observe that, when $\ell = 1$, all instances are solved to optimality within our proposed time limit. We also observe that the CPU time for solving (4.2) increases as ℓ increases. Although (4.2) contains fewer variables for larger values of ℓ, the branch-and-bound tree grows significantly as ℓ increases, as observed in Table 4.1. Thus, (4.2) reaches the time limit in three out of five instances when $\ell = 5$. The average optimality gap at the time limit for these three instances is about 1.5%.

Table 4.1: Average MFP-S results over networks having 10 relay nodes

	CPU Time (sec.)		# of Branches		# Solved		Opt. Gap		# of Columns
Lower bound ℓ	MIP	B&P	MIP	B&P	MIP	B&P	MIP	B&P	B&P
1	892.9	0.3	46835.0	1	5	5	0%	0%	20.0
3	1794.3	0.9	95415.6	13.0	4	5	0.3%	0%	32.6
5	2461.4	1.0	226250.2	5.8	2	5	0.9%	0%	44.4

For networks having 15 relay nodes, Table 4.2 shows that B&P requires, on average, less than 13 seconds to terminate with an optimal solution. (The maximum time to solve any instance was 46 seconds.) As observed in Table 4.1, there is an increase in CPU time when increasing ℓ from 1 to 3, but only a slight CPU time increase when ℓ is increased to 5. Among all instances, B&P found three arc-stable solutions that were not static-stable. For two of these instances, our SPDP algorithm was able to obtain a static-stable solution without branching directly on a λ-variable. In the third instance (for which $\ell = 5$), our algorithm was forced to branch on a λ-variable. While the CPU time for this instance was only 14.9 seconds, the time to solve the same instance for ℓ-values equal to 1 and 3 was 0.6 and 1.4 seconds, respectively. As expected, branching on a λ-variable increased the relative CPU time for solving this instance.

By contrast, model (4.2) solved the MFP-S to optimality in only three total instances, each having $\ell = 1$. Unlike networks having 10 relay nodes, model (4.2) reached the memory limit in three instances. For ℓ-values equal to 3 or 5, the average optimality gap at termination was about 3%. As expected, even when (4.2) found an optimal solution within the time or memory limits, the B&P algorithm required significantly less computational time for these networks.

Table 4.2: Average MFP-S results over networks having 15 relay nodes

	CPU Time		# of Branches		# Solved		Opt. Gap		# of Columns
Lower bound ℓ	MIP	B&P	MIP	B&P	MIP	B&P	MIP	B&P	B&P
1	2447.5	3.2	87709.6	45.8	3	5	0.4%	0%	104.2
3	2953.7	11.8	152976.4	68.2	0	5	3.8%	0%	177.8
5	3570.3	12.6	439615.0	154	0	5	3.8%	0%	183.6

Finally, Table 4.3 displays the computational results for solving the MFP-S on networks having 20 relay nodes. For two instances, the B&P algorithm invoked the SPDP subroutine to successfully obtain a static-stable solution without branching on a λ-variable. Similar to the results for smaller networks, the B&P outperforms model (4.2). Model (4.2) reached the time or memory limit in all instances, and the average optimality gap at termination was at least 10% for all values of ℓ. (For each instance that terminates due to memory limitations, we record the CPU time of the instance as the time at which the memory limit was reached. Hence, the average CPU time reported for the model (4.2) when $\ell = 3$ is less than 3600, since some of those instances reached the memory limit.)

On the same set of instances, we note that the average CPU time for the B&P actually decreased as ℓ increased from 3 to 5 for those instances. We suspect that, in most of the instances for which $\ell = 5$, heuristics used by SCIP were able to cut off a large number of infeasible solutions before entering the branching phase. In doing so, the resulting branching tree was smaller. Thus, increasing ℓ seems to have a minor effect on the computational time of the B&P.

Table 4.3: Average MFP-S results over networks having 20 relay nodes

	CPU Time		# of Branches		# Solved		Opt. Gap		# of Columns
Lower bound ℓ	MIP	B&P	MIP	B&P	MIP	B&P	MIP	B&P	B&P
1	3600.0	3.7	1232.8	24.2	0	5	29.4%	0%	77.2
3	2749.3	9.8	15337.8	99.2	0	5	10.8%	0%	203.8
5	3600.0	4.1	120627.2	34.4	0	5	14.0%	0%	207.2

4.3.2 Scalability of B&P Algorithm

Next, we examine the scalability of our B&P approach. To do so, we solve five randomly-generated MFP-S problem instances of networks having 25, 30, 35, 40, 45, and 50 relay nodes. In this section, we present the average CPU time, average number of branches in the branching tree, the number of instances solved to optimality, and the average number of generated columns. We set the optimality gap to 0.1%, the time limit to 3600 seconds, and let $\ell = 3$ for all instances.

Table 4.4 shows that as the size of the network increases, the CPU time, the size of the branching tree, and the number of generated columns also increase. On average, networks having 25 relay nodes were solved within 11 seconds while networks having 30 relay nodes were solved in about 32 seconds. It is worth noting that for all networks having 35 relay nodes or fewer, the B&P algorithm solved all five instances to optimality within the allotted time and memory limits. For networks having 40 or 45 nodes, two out of five instances were solved to optimality before reaching the memory limit. The average optimality gap after reaching the memory limit in these three instances was about 0.3%. Thus, even after reaching the memory limit, B&P provided a relatively high-quality lower bound on the MFP-S. For networks having 50 nodes, the B&P reached the memory limit quicker than network instances having 40 or 45 relay nodes, due to the large size of the master problem and associated branching tree.

Table 4.4: Average MFP-S results using the B&P approach where $\ell = 3$

Number of nodes	CPU Time	# of Branches	# Solved	Opt. Gap	# of Columns
25	10.9	73.8	5	0%	249.4
30	32.0	201.6	5	0%	436.0
35	53.6	215.6	5	0%	762.0
40	142.1	291.4	2	0.7%	1004.2
45	119.4	414.4	2	0.4%	1080.6
50	96.7	128.2	0	1.1%	1008.4

Chapter 5

Dynamic Network Flow Problems in Non-simultaneous Flow Settings

Network flow problems commonly employ static flow assumptions in which flows are simultaneously transmitted. Alternatively, some applications require dynamic flows to be transmitted according to a non-simultaneous schedule. In dynamic flow settings, an arc may be required to undergo setup each time it begins transmitting flow. Such setup requirements may be modeled in terms of flow. For example, reconsider the machine scheduling problem discussed in Chapter 1 in which the flow on a path refers to the amount of time a worker utilizes a machine to process a job. A worker likely requires some preparation time after moving from one machine to another. In this case a policy might require each worker to spend at least some minimum amount of time processing jobs on a machine before moving to another machine. Scheduling a worker on a machine for some time less than this preparation time is often undesirable for many reasons, including safety and quality considerations that arise when workers switch tasks (see, e.g., [Monsell, 2003]).

The problems we consider in this chapter take place on a graph $\mathcal{G}$ consisting of node set $\mathcal{V}$ and arc set $\mathcal{A}$. Set $\mathcal{V}$ contains a single source s, a single sink t, and a set of intermediate nodes. Each arc $(i,j) \in \mathcal{A}$ has a capacity of $c_{ij} > 0$, which refers to the maximum amount of flow that can be sent on (i,j). We assume that non-simultaneous flows can only be sent on one s–t path at a time. Thus, a network flow must be decomposed into a set of paths, which can in turn be organized into a schedule $\mathcal{P}$ corresponding to the order in which flow is sent across each path. Schedule $\mathcal{P}$ is composed of paths $p_1, \ldots, p_q$ having flow values $f_1, \ldots, f_q$, where $f_k > 0,\ \forall k = 1, \ldots, q$, and that flow is transmitted on the paths in that order.

There are many dynamic flow applications in addition to the machine scheduling setting discussed in Chapter 1. Consider the presence of setup costs, in which a fixed cost is incurred each time an arc begins transmission. In WSN optimization, operators are required to pay a fixed cost to begin transmitting information from one sensor to another [Yick et al., 2008]. These fixed costs often correspond to the financial and/or computational effort required to establish a secure communication link between a pair of sensors [Perrig et al., 2004, Shi and Perrig, 2004]. Whenever a sensor discontinues transmission, this communication link either ceases to exist or is assumed not to be secure. Thus, each time a sensor begins transmission, a new secure communication link must be set up to avoid interference.

We study a pair of dynamic flow network flow problems, in which the first considers the presence of setup constraints, and the second considers the presence of setup costs. The former problem can be modeled by the maximum flow problem having dynamic stability restrictions (MFP-D). The latter problem can be modeled by the minimum-cost flow problem having *arc-activation* costs (MCF-A), in which an arc (i, j) is said to be *activated* on path p when (i, j) has positive flow on the p^{th} scheduled path, but not on the $(p-1)^{st}$ scheduled path.

Our contributions in this chapter are as follows. One, we discuss the notion of dynamic stability and present a mixed-integer programming (MIP) model for the MFP-D. Two, we present a heuristic algorithm that obtains lower and upper bounds for the MFP-D. Three, we provide motivation for the MCF-A and present an MIP model for its solution. Four, as an alternative to this MIP, we employ a relaxation-based algorithm for obtaining upper and lower bounds.

In Section 5.1.1 we describe the MFP-D, and we present an MIP model for its solution in Section 5.1.2. Section 5.1.3 details our heuristic algorithm for the MFP-D, and Section 5.1.4 details computational results for displaying the performance of this heuristic. Section 5.2 describes the MCF-A and its applications, and Section 5.2.1 presents an MIP model for the MCF-A. Section 5.2.3 details our bounding algorithm for the MCF-A. Section 5.2.4 finally details the efficacy of this algorithm when compared to an MIP.

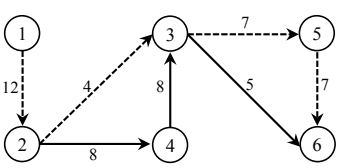
5.1 Dynamic Flow Stability

In this section we consider stability restrictions under the dynamic flow assumptions. In Section 5.1.1, we describe the MFP with dynamic-stable restrictions on paths, and compare it to

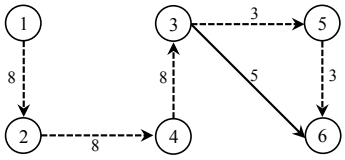
problems in foregoing chapters of this work. Then, in Section 5.1.2 we give an MIP formulation for the MFP-D. Finally, we present a series of heuristic algorithms to determine lower and upper bounds for the MFP-D in Section 5.1.3.

5.1.1 Problem Description and Model

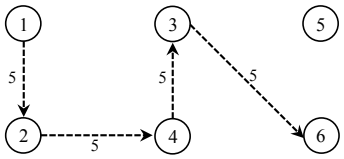
To first explain the notion of dynamic stability, a schedule S of paths is said to be feasible to the MFP-D if for every arc $(i, j) \in \mathcal{A}$, we have $\sum_{k=t}^{v} f_k \geq \ell$ if (i, j) belongs to all paths $p_t, p_{t+1}, \ldots, p_v$, but not p_{t-1} or p_{v+1}, where p_0 and p_{q+1} are defined as empty paths. A network flow solution is dynamic stable if some ordered path decomposition exists that satisfies the above restrictions. The motivation behind dynamic stability is that when an arc is used to send flow, it sends at least ℓ units of flow in an uninterrupted interval of time. While the dynamic-stable lower bound can be arc-dependent, we assume a uniform lower bound ℓ for all arcs in $\mathcal{A}$ for simplicity of notation.



(a) Example flow with path p_1 (dashed)



(b) Path p_2 (dashed) and updated remaining flows after p_1



(c) Path p_3 (dashed) and updated remaining flows after p_2

Figure 5.1: Dynamic-stable example with $\ell = 4$

To illustrate the notion of dynamic stability, consider the flows depicted in Figure 5.1a, and suppose that $\ell = 4$. To determine if there exists a schedule of paths for these flows that meets dynamic-stable restrictions, we first note that such a schedule must include some path that sends 4 units of flow on arc $(2, 3)$. We let this path be $p_1 = (1, 2, 3, 5, 6)$. After flow is sent on p_1, arcs $(3, 5)$ and $(5, 6)$ both have 3 remaining units of flow to send. These arcs must be included in the next path p_2 to remain stable. If they are not included in p_2, some future path would need to send the remaining $3 < \ell$ units of flow along these arcs.

The next path in the schedule must therefore be $p_2 = (1, 2, 4, 3, 5, 6)$, which has 3 units of

flow. The last path must be $p_3 = (1, 2, 4, 3, 6)$, having 5 units of flow, which completes the path-flow schedule. Figure 5.2 displays a Gantt chart depicting time intervals during which arcs send flow (normalized so that transmitting each unit of flow corresponds to one unit of time). Note that each such interval is no less than 4 units. Thus, while this set of paths is dynamic stable, there does not exist any static-stable solution corresponding to the flows depicted in Figure 5.1a.

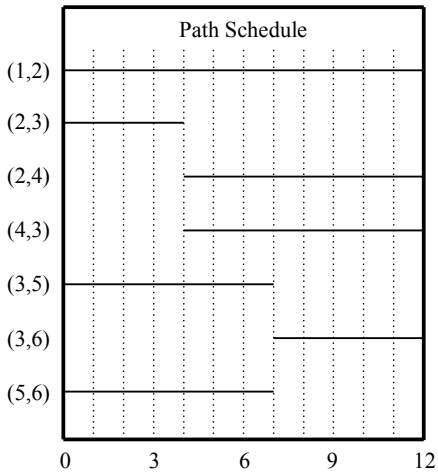


Figure 5.2: Gantt chart of arc flows corresponding to flows in Figure 5.1

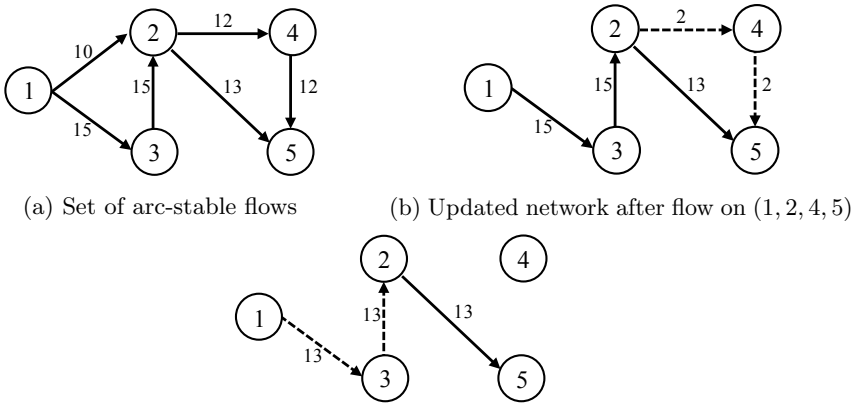
For the MFP-D, an optimal set of flows must be organized into an ordered schedule of path flows. The dynamic-stable restrictions state that the sum of flows on successive positive-flow paths using each arc $(i, j) \in \mathcal{A}$ must be at least ℓ. A schedule $\mathcal{P}$ of flows $\bar{x}$ is said to be aggregate arc stable when $\sum_{p=1}^{|\mathcal{P}|} \bar{x}_{ij} \in \{0, [\ell, c_{ij}]\},\ \forall (i, j) \in \mathcal{A}$, in which arc capacity c_{ij} corresponds to an upper bound on x_{ij}. As described in a previous chapter an *arc-stable* solution is one for which $x_{ij} \in \{0, [\ell, c_{ij}]\},\ \forall (i, j) \in \mathcal{A}$.

Lemma 1. *Any feasible dynamic-stable solution is also an aggregate arc-stable solution.*

Proof. Consider a set of dynamic-stable flows. For each $(i, j) \in \mathcal{A}$, there exists at least one subset $\{k, k+1, \ldots, v\} \subseteq S$ of ordered paths having flows $\{f_k, f_{k+1}, \ldots, f_v\}$ on (i, j) in which $\sum_{i=k}^{v} f_i \geq \ell$. Thus, the aggregate flow on (i, j) among all paths in S is also greater than or equal to ℓ. □

However, arc stability does not necessarily imply dynamic stability. Appendix C describes an example arc-stable solution to show this implication. Furthermore, a dynamic-stable solution is not necessarily a static-stable solution either. Recall from Chapter 4 that the flows in Figure 5.3a cannot be decomposed into a static-stable solution. We show that there does exist a dynamic-stable

solution corresponding to these flows. Let the schedule begin with path $(1, 2, 4, 5)$ having 10 units of



(a) Set of arc-stable flows

(b) Updated network after flow on $(1, 2, 4, 5)$

(c) Updated network after flow on $(1, 3, 2, 4, 5)$

Figure 5.3: Network example with $\ell = 10$

flow. Arcs $(2, 4)$ and $(4, 5)$ are active in the updated network (Figure 5.3b). The next path must be $(1, 3, 2, 4, 5)$ having 2 units of flow. The last path must then be $(1, 3, 2, 5)$ having 13 units of flow. Therefore, the arc flows in Figure 5.3a correspond to a dynamic-stable solution, but not a static-stable solution. For the alternative implication, consider Lemma 2.

Lemma 2. *Any feasible static-stable solution is also a dynamic-stable solution.*

Proof. Consider a static-stable solution. In this case, the flow f_k on each positive-flow path $p_k, \forall k \in S$ is greater than or equal to ℓ. Any ordering of these paths in the dynamic-stable problem is therefore feasible. □

Therefore, the set of arc-stable solutions contains the set of dynamic-stable solutions, which in turn contains the set of static-stable solutions.

5.1.2 MFP-D MIP Formulation

We now present an MIP model for the MFP-D. Let $\mathcal{S} = \{1, \ldots, |\mathcal{A}|\}$ be the index set for the path schedule solution. (Our choice of $|\mathcal{A}|$ is chosen as the maximum number of paths in any schedule; as before, computing a precise bound is beyond the scope of this work.) Let the set Ω contain all combinations of $u, v \in S$ such that $u \leq v$. Variable x_{ij}^p represents the flow on arc (i, j) for the p^{th} scheduled path. Let y_{ij}^p be a binary variable that equals 1 if arc (i, j) is on the p^{th} scheduled path and 0 otherwise. The variable w_{ij}^{uv} equals 1 if arc (i, j) sends flow on the set of consecutive

paths $u, \ldots, v$ such that $(u, v) \in \Omega$, and 0 otherwise. The MFP-D can be modeled as follows.

$$\max\ z \tag{5.1a}$$

$$\text{s.t.} \sum_{p\in\mathcal{S}} \sum_{i:(s,i)\in\mathcal{A}} x^p_{si} - z = 0 \tag{5.1b}$$

$$-\sum_{p\in\mathcal{S}} \sum_{j:(j,t)\in\mathcal{A}} x^p_{jt} + z = 0 \tag{5.1c}$$

$$\sum_{p\in\mathcal{S}} \sum_{j:(i,j)\in\mathcal{A}} g_{ij} x^p_{ij} \le b_i \quad \forall i \in \mathcal{V} \tag{5.1d}$$

$$x^p_{ij} \le c_{ij} y^p_{ij} \quad \forall (i,j) \in \mathcal{A},\ \forall p \in \mathcal{S} \tag{5.1e}$$

$$\sum_{j:(i,j)\in\mathcal{A}} y^p_{ij} - \sum_{h:(h,i)\in\mathcal{A}} y^p_{hi} = 0 \quad \forall i \in \mathcal{V}\setminus\{s,t\},\ p \in \mathcal{S} \tag{5.1f}$$

$$x^p_{ij} = \phi_p y^p_{ij} \quad \forall p \in \mathcal{S},\ (i,j) \in \mathcal{A} \tag{5.1g}$$

$$\sum_{i:(s,i)\in\mathcal{A}} y^1_{si} = 1 \tag{5.1h}$$

$$\sum_{i:(s,i)\in\mathcal{A}} y^p_{si} \le \sum_{i:(s,i)\in\mathcal{A}} y^{p-1}_{si} \quad \forall p = 2, \ldots, |\mathcal{S}| \tag{5.1i}$$

$$w^{uv}_{ij} \le y^p_{ij} \quad \forall (i,j) \in \mathcal{A},\ (u,v) \in \Omega,\ p = u, \ldots, v \tag{5.1j}$$

$$w^{uv}_{ij} \ge \sum_{p=u}^{v} y^p_{ij} - (v-u) \quad \forall (i,j) \in \mathcal{A},\ (u,v) \in \Omega \tag{5.1k}$$

$$\ell\left(w^{uv}_{ij} - w^{u-1,v}_{ij} - w^{u,v+1}_{ij}\right) \le \sum_{p=u}^{v} x^p_{ij} \quad \forall (i,j) \in \mathcal{A},\ (u,v) \in \Omega \tag{5.1l}$$

$$\phi_p \ge 0 \quad \forall p \in \mathcal{S} \tag{5.1m}$$

$$w^{uv}_{ij} \ge 0 \quad \forall (i,j) \in \mathcal{A},\ (u,v) \in \Omega \tag{5.1n}$$

$$y^p_{ij} \in \{0,1\} \quad \forall (i,j) \in \mathcal{A},\ p \in \mathcal{S} \tag{5.1o}$$

Objective function (5.1a) and constraints (5.1b) and (5.1c) maximize the amount of flow z transmitted from s to t. Constraints (5.1d) and (5.1e) enforce node- and arc-capacity restrictions, respectively. Constraints (5.1f) enforce flow balance at all intermediate nodes, and constraints (5.1g) enforce ϕ_p to be the flow on each path $p \in S$. Constraints (5.1h) and (5.1i) are a set of symmetry-breaking constraints. Constraints (5.1j) and (5.1k) ensure that variable w^{uv}_{ij} equals 1 if and only if consecutive paths from u to v have positive flow and contain arc (i, j). Constraints (5.1l) enforce that the sum of consecutive paths from u to v using arc (i, j) must be 0 or at least as large as ℓ. Constraints

(5.1m) and (5.1n) enforce non-negativity restrictions on the ϕ- and w-variables and constraints (5.1o) enforce binary restrictions on y.

Observe that in the MFP-S, any positive-flow subtours can be removed *a posteriori* from a solution without affecting its feasibility or its objective function value, given that $g_{ij} \geq 0,\ \forall (i,j) \in \mathcal{A}$. Model (5.1) does not prevent positive-flow subtours, because removing a positive-flow subtour from an MFP-D solution may cause that solution to become infeasible. For example, consider a schedule of paths from a solution to (5.1) for which arc (i,j) has positive flow on a subtour contained within p_k. Suppose that (i,j) also has positive flow on p_{k-1} but not p_{k-2} or p_{k+1}. If the flow on p_{k-1} is less than ℓ and the flow on the subtour in p_k is removed, then the revised solution would not be dynamic stable with respect to the flow on (i,j). Subtours can thus play the role of allowing (otherwise unnecessary) flows to be transmitted for the purpose of meeting dynamic-stable restrictions.

For some applications flows on these subtours can actually be routed, and so model (5.1) is a valid formulation. If subtours need to be prohibited, then we can modify (5.1) by introducing additional variables $\mu_i^p,\ \forall p = 1, \ldots, |S|,\ i = |\mathcal{V}| \setminus \{s,t\}$ that represent the position of node i along the interior of path p (not including nodes s or t). Define $\mathcal{A}^{\mathrm{I}}$ as the set of *interior arcs*, defined as $(i,j) \in \mathcal{A} : i \neq s$ and $j \neq t$. The following Miller-Tucker-Zemlin [Miller et al., 1960] (MTZ) constraints eliminate subtours:

$$\mu_j^p \geq \mu_i^p + 1 + (|\mathcal{V}| - 2)(y_{ij}^p - 1) \quad \forall p = 1, \ldots, |S|,\ \forall (i,j) \in \mathcal{A}^{\mathrm{I}} \tag{5.2a}$$

$$0 \leq \mu_i^p \leq (|\mathcal{V}| - 3) \sum_{j:(i,j)\in\mathcal{A}} y_{ij}^p \quad \forall p = 1, \ldots, |S|,\ \forall i \in \mathcal{V} \setminus \{s,t\}. \tag{5.2b}$$

Constraints (5.2a) force $\mu_j^p \geq \mu_i^p + 1$ when $y_{ij}^p = 1,\ \forall (i,j) \in \mathcal{A}^{\mathrm{I}}$, while constraints (5.2b) ensure that μ_i^p is positive only when a path visits node i, and otherwise bounds the μ-variables between 0 and $|\mathcal{V}| - 3$.

A column generation approach to solving the MFP-D may not be a promising approach since it requires an ordering of such paths, as opposed to an unordered collection in simultaneous flow setting. Furthermore, determining valid inequalities to cut off infeasible solutions to (5.1) is difficult due to interdependencies within the path schedule.

5.1.3 MFP-D Heuristic Algorithm

As described in Section 5.1.1, an *arc-stable* solution, in which $\sum_{p \in S} x_{ij}^p \in \{0, [\ell, c_{ij}]\}$, yields an upper bound on the MFP-D. We can find such a solution either by solving a maximum flow problem formulation with constraints $uy_{ij} \geq x_{ij} \geq \ell y_{ij}$, $\forall (i,j) \in \mathcal{A}$, as in [Beale, 1985], or by executing the B&P algorithm using only the branching mechanism described in Chapter 4. Additionally, any feasible solution to the MFP-D yields a lower bound. We thus present a heuristic that takes an arc-stable solution as input and attempts to create a dynamic-stable solution whose aggregate flows are similar to those of the arc-stable solution. This process thus yields lower and upper bounds on the optimal MFP-D objective and assumes that positive-flow subtours are allowed in a dynamic-stable solution.

We first define F_{ij} as the flow on $(i,j) \in \mathcal{A}$ in the given arc-stable solution and $\bar{F}$ as the maximum flow value obtained from our heuristic. At each iteration k of our heuristic, we will determine a new path and its associated flow to include in the solution. The heuristic defines remaining flows, $r_{ij}^k \geq 0$, as F_{ij} minus the flow transmitted on arc $(i,j) \in \mathcal{A}$ on the first $k-1$ paths identified by the heuristic. Also, at iteration k, $\mathcal{A}^k$ denotes the set of active arcs that must be included in the k^{th} path for the final schedule to be dynamic stable. The MFP-D heuristic is given as follows.

Initialization: Set $k = 1$, $\mathcal{A}^1 = \emptyset$, and $r_{ij}^1 = F_{ij}$, $\forall (i,j) \in \mathcal{A}$. Set $\bar{F} = 0$ and let the incumbent solution be null.

Step 1: If $r_{ij}^k = 0$, $\forall (i,j) \in \mathcal{A}$, then go to Step 4b. Otherwise, let $r_{min}^k = \min\{r_{ij}^k : (i,j) \in \mathcal{A},\ r_{ij}^k > 0\}$. If $\mathcal{A}^k = \emptyset$, then go to Step 2a, and otherwise go to Step 2b.

Step 2a: Find an s–t path p_k consisting only of arcs $(i,j) \in \mathcal{A} : r_{ij}^k > 0$, such that $r_{ij}^k = r_{min}^k$ for at least one arc (i,j) belonging to p_k. Appendix C describes how this path is computed. Assign a flow of $f_k = r_{min}^k$ to path p_k and go to Step 3.

Step 2b: Find any s–t path p_k consisting only of arcs $(i,j) \in \mathcal{A} : r_{ij}^k > 0$ that contains all arcs in $\mathcal{A}^k$, as detailed in Appendix C. Set f_k equal to the minimum r_{ij}^k-value among all arcs belonging to p_k. If such a path does not exist, then go to Step 4a. Otherwise, continue to Step 3.

Step 3: For each arc (i,j) on path p_k, set $r_{ij}^{k+1} = r_{ij}^k - f_k$; for all other arcs (i,j), set $r_{ij}^{k+1} = r_{ij}^k$. Place all active arcs in $\mathcal{A}^{k+1}$. Attempt to convert this partial solution into a feasible solution by executing the *improvement phase* discussed below. Iterate by setting $k = k+1$, and return to Step 1.

Step 4a: Terminate the algorithm and report the incumbent solution. The value $\bar{F}$ yields a lower bound on MFP-D.

Step 4b: Terminate the algorithm. The current schedule of paths $p_1, p_2, \ldots, p_{k-1}$ with corresponding flows $f_1, f_2, \ldots, f_{k-1}$ is dynamic stable. Moreover, the flow obtained in this solution equals that of the given arc-stable solution, and hence the heuristic solution must be optimal to the MFP-D.

At Step 3 of the foregoing algorithm, we attempt to find a feasible MFP-D solution to strengthen the MFP-D lower bound via an improvement phase. This phase operates by appending path flows to the end of a partial solution obtained so far in the algorithm, with the goal of identifying a good-quality feasible solution. The appended paths are temporary in the sense that they are not directly used in the remainder of the main algorithm. The improvement phase steps are described as follows.

Improvement Step 1: If $\mathcal{A}^{k+1}$ contains active-nonstable arcs (see Section 5.1.1), then go to Improvement Step 2. Otherwise, the set of paths $p_1, p_2, \ldots, p_k$ having respective flows $f_1, f_2, \ldots, f_k$ corresponds to a feasible MFP-D solution. Insert a dummy null path $\bar{p}_{k+1}$ having flow $\bar{f}_{k+1} = 0$, and go to Improvement Step 3.

Improvement Step 2: Find a path $\bar{p}_{k+1}$ that includes all active-nonstable arcs in $\mathcal{A}^{k+1}$. The method for finding this path is described in Appendix C. If a path $\bar{p}_{k+1}$ is found and $\bar{f}_{k+1}$ results in all active-nonstable arcs becoming either active-stable or not active, then go to Improvement Step 3. Otherwise, exit the improvement phase and continue with the heuristic.

Improvement Step 3: A feasible solution has been identified. If $\sum_{g=1}^{k} f_g + \bar{f}_{k+1} > \bar{F}$, then set $\bar{F} = \sum_{g=1}^{k} f_g + \bar{f}_{k+1}$ and update the incumbent solution accordingly. Set counter $h = 2$ and continue to Improvement Step 4.

Improvement Step 4: Find a static-stable s–t path $\bar{p}_{k+h}$ that is feasible with respect to the remaining node and arc capacities after flows $f_1, f_2, \ldots, f_k, \bar{f}_{k+1}, \bar{f}_{k+2}, \ldots, \bar{f}_{k+(h-1)}$ are transmitted.

Appendix C describes how to find such a path. If a path $\bar{p}_{k+h}$ exists, then go to Improvement Step 5. If no such path exists, then terminate the improvement phase and continue with the heuristic.

Improvement Step 5: Determine the largest possible flow value $\bar{f}_{k+h}$ on path $\bar{p}_{k+h}$ without violating any node or arc capacities. If $\sum_{g=1}^{k} f_g + \sum_{i=1}^{h} \bar{f}_{k+i} > \bar{F}$, set $\bar{F} = \sum_{g=1}^{k} f_g + \sum_{i=1}^{h} \bar{f}_{k+i}$ and update the incumbent. Set $h = h + 1$ and repeat Improvement Step 4.

5.1.4 Computational Results

In this section, we explore the trade-off between implementing our B&P approach in Chapter 4 and solving model (5.1) when seeking an optimal solution to the MFP-D. While using the B&P algorithm is only guaranteed to obtain a lower bound on the MFP-D, model (5.1) is much larger and more difficult to solve. Thus, we explore whether the possible improvement in the objective function value by solving (5.1) is worth the additional computational effort in doing so. Thus, we solve the MFP-S and the MFP-D via our B&P approach and model (5.1), respectively, for ten randomly-generated networks having only seven relay nodes. For all instances, we set $\ell = 1$ and use the same ranges of b, g, and c as before in Chapter 4, in which $b_i \in [25, 100],\ \forall i \in \mathcal{V}$ and $g_{ij} \in [1, 12]$ and $c_{ij} \in [10, 20],\ \forall (i, j) \in \mathcal{A}$. A preliminary investigation revealed that model (5.1) was unable find an optimal solution for most instances before reaching the time or memory limits for networks having more than seven relay nodes, or for instances having $\ell > 1$.

Table 5.1 displays the objective function values and CPU times for solving the MFP-S and MFP-D via the B&P approach and model (5.1) for each instance. For nine of the ten instances, both approaches found the same objective function value, but model (5.1) took significantly longer to solve. Thus, the static-stable solution provided by the B&P approach was also dynamic-stable in all of these instances. The average CPU time for the B&P approach was about 0.4 seconds. For those instances solved within the time limit, the average CPU time to solve the MFP-D was about 356 seconds. Thus, the B&P approach obtains an arc-stable solution that provides a relatively high-quality lower bound on the MFP-D much faster than (5.1) obtains an optimal dynamic-stable solution. In almost all instances, this lower bound also corresponds to an optimal dynamic-stable solution.

Next, we examine the performance of our MFP-D heuristic presented in Section 5.1.3 for obtaining lower and upper bounds on the MFP-D. To do so, we execute this heuristic starting from the arc-stable solutions provided by the B&P approach on the fifteen network instances having 20

Table 5.1: Results for solving the MFP-S and the MFP-D

Instance	Objective Function Value		CPU Time	
	MFP-S	MFP-D	MFP-S	MFP-D
1	31.3	31.3	0.2	92.7
2	30.5	30.5	0.2	129.7
3	36.3	36.3	0.2	72.7
4	22.2	22.2	0.2	93.4
5	28.7	28.7	0.8	3600
6	32.3	32.3	0.2	3600
7	32.5	32.5	0.2	104.9
8	39.6	39.6	0.3	98.5
9	40.2	40.2	1.0	48.0
10	34.5	34.5	0.6	2208.6

relay nodes described in Chapter 4. Table 5.2 displays the results of executing our heuristic on these flow solutions. For all instances, the MFP-D heuristic was able to successfully decompose the given arc-stable solution into a dynamic-stable solution in less than 0.01 seconds, resulting in a provably optimal solution for each instance over each $\ell \in \{1, 3, 5\}$. When coupled with the B&P approach to obtain an arc-stable solution, the average CPU time was 3.8s, 9.9s, and 5.4s when $\ell = 1$, 3, and 5, respectively.

Table 5.2: Average results for the MFP-D heuristic for networks having 20 relay nodes

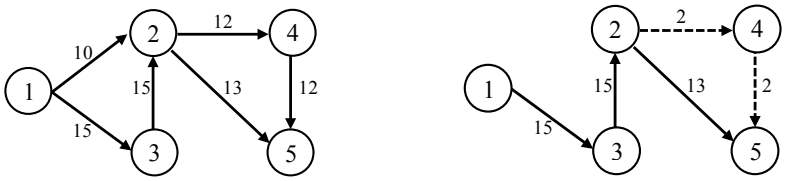
Lower bound ℓ	Gap between LB and UB	CPU Time
1	0%	3.8
3	0%	9.9
5	0%	5.4

5.2 Minimum-cost Flow Problem Having Arc-activation Costs

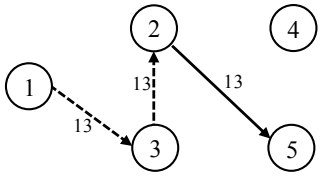
In this section we consider the presence of arc-activation costs in minimum-cost network flow problems in dynamic flow settings. In Section 5.2.1, we describe the MCF-A, its applications, and its relationship to the fixed-charge network flow problem. We next present an MIP model for the MCF-A in Section 5.2.2, and Section 5.2.3 decomposes this MIP to improves its solvability and details an algorithm for obtaining upper and lower bounds for the MCF-A. Finally, we present computational results detailing the effectiveness of our algorithm in Section 5.2.4.

5.2.1 Introduction and Problem Statement

For each arc $(i, j) \in \mathcal{A}$, define $a_{ij} > 0$ as the cost to activate arc $(i, j) \in \mathcal{A}$, and define $d_{ij} > 0$ as the cost to transmit a unit of flow on arc $(i, j) \in \mathcal{A}$. The MCF-A is defined as the problem of determining a schedule of path flows $\mathcal{P}$ that minimize the sum of the activation and transmission costs required to send f units of flow from s to t. The MCF-A stipulates that activation cost a_{ij} is incurred each time a sequence of paths in $\mathcal{P}$ begins transmitting flow on (i, j). For example, recall from Section 5.1.1 the following flows in Figure 5.4a.



(a) Sum of flows in an MCF-A solution (b) Updated network after flow on $(1, 2, 4, 5)$



(c) Updated network after flow on $(1, 3, 2, 4, 5)$

Figure 5.4: Network flow example for an MCF-A solution

The sequence of paths in $\mathcal{P}$ is $p_1 = (1, 2, 4, 5)$, $p_2 = (1, 3, 2, 4, 5)$, and $p_3 = (1, 3, 2, 5)$ having 10, 2, and 13 units of flow, respectively. Path p_1 activates arcs $(1, 2)$, $(2, 4)$, and $(4, 5)$. Next, path p_2 activates arcs $(1, 3)$ and $(3, 2)$ since $(2, 4)$ and $(4, 5)$ were included on the previously scheduled path p_1. Finally, path p_3 only activates $(2, 5)$. Thus, $\mathcal{P}$ activates each positive-flow arc only once, yielding a total activation cost of $a_{12} + a_{24} + a_{45} + a_{13} + a_{32} + a_{25}$. Alternatively, consider the schedule of paths $\mathcal{P}' = \{p_1, p_3, p_2\}$ having 10, 13, and 2 units of flow, respectively. Schedule $\mathcal{P}'$ incurs a total activation cost of $a_{12} + 2a_{24} + 2a_{45} + a_{13} + a_{32} + a_{25}$ since both paths p_1 and p_2 activate arcs $(2, 4)$ and $(4, 5)$.

The MCF-A is closely related to the fixed-charge network flow problem (FCNFP), in which any positive flow on an arc incurs a single fixed cost in addition to any transmission costs [Hochbaum and Segev, 1989]. The MCF-A is not equivalent to the FCNFP, since an optimal MCF-A solution may require an arc to be activated multiple times. The flows in Figure 5.5 display a case in which

this occurs. The flows in Figure 5.5 must be decomposed into a schedule of paths $\mathcal{P}$ to determine

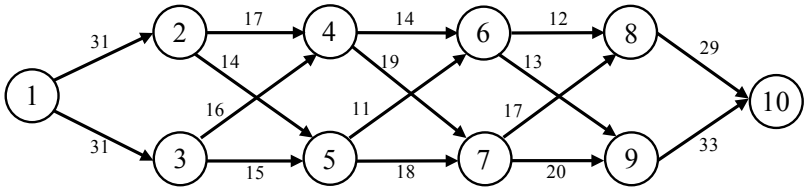


Figure 5.5: MCF-A solution in which arcs must be activated more than once

if an MCF-A solution exists that activates all arcs only once. Consider when $\mathcal{P}$ begins with path $p_1 = (1, 2, 4, 6, 8, 10)$ having 12 units of flows, such that the flow on arc $(6, 8)$ is exhausted. Path p_1 thus activates arcs $(1, 2),\ (2, 4),\ (4, 6),\ (6, 8)$, and $(8, 10)$ for the first time. The next path in $\mathcal{P}$ must include arcs $(1, 2),\ (2, 4),\ (4, 6)$, and $(8, 10)$ to send flows on cycles within the network, thus avoiding the reactivation of certain arcs. Since no other positive-flow paths from 1 to 10 exist that contain both arcs $(4, 6)$ and $(8, 10)$, at least one of these arcs must be activated again on a later path in $\mathcal{P}$. Further examination shows that some arcs must be activated multiple times for any MCF-A solution regardless of which path starts $\mathcal{P}$.

As described in Chapter 1, the MCF-A can be used to model network flow problems in interdependent infrastructure systems. Cavdaroglu et al. [Cavdaroglu et al., 2013] study the problem of restoring public services after a disaster disrupts civil infrastructure systems. In these problems, the managers are required to pay a fixed financial or labor cost to deploy temporary arcs that provide an interim infrastructure for deploying essential and non-essential services (e.g. transportation, telecommunication, and power). Restoration and planning decisions must be made according to non-simultaneous schedule since these infrastructure are commonly interdependent. Therefore, the MCF-A could be employed to model a variation of this problem, in which arcs expire after infrastructures are deconstructed.

5.2.2 MCF-A MIP Model

For the MCF-A, define a schedule of paths $\mathcal{P} = \{1, \ldots, f\}$. (Note that f is a valid bound on the size of $\mathcal{P}$ because an optimal solution exists in which all flows are integer-valued.) Continuous variable x_{ij}^p refers to the amount of flow on arc $(i, j) \in \mathcal{A}$ on the p^{th} path in $\mathcal{P}$, binary variable y_{ij}^p equals 1 if and only if arc $(i, j) \in \mathcal{A}$ is activated on the p^{th} path, and binary variable z_{ij}^p equals 1 if and only if arc $(i, j) \in \mathcal{A}$ has positive flow on the p^{th} path. Defining $z_{ij}^0 = 0,\ \forall (i, j) \in \mathcal{A}$, the MCF-A

can be modeled by the following MIP.

$$\min \sum_{p=1}^{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{A}} \left(d_{ij}x_{ij}^p + a_{ij}y_{ij}^p\right) \tag{5.3a}$$

$$\text{s.t.} \sum_{p=1}^{|\mathcal{P}|} \sum_{i:(s,i)\in\mathcal{A}} x_{si}^p = f, \tag{5.3b}$$

$$\sum_{j:(i,j)\in\mathcal{A}} x_{ij}^p - \sum_{h:(h,i)\in\mathcal{A}} x_{hi}^p = 0 \quad \forall i \in \mathcal{V}\setminus\{s,t\},\ p=1,\ldots,|\mathcal{P}|, \tag{5.3c}$$

$$\sum_{i:(s,i)\in\mathcal{A}} z_{si}^p \le 1 \quad \forall p=1,\ldots,|\mathcal{P}|, \tag{5.3d}$$

$$\sum_{j:(i,j)\in\mathcal{A}} z_{ij}^p - \sum_{h:(h,i)\in\mathcal{A}} z_{hi}^p = 0 \quad \forall i \in \mathcal{V}\setminus\{s,t\},\ p=1,\ldots,|\mathcal{P}|, \tag{5.3e}$$

$$z_{ij}^p \le x_{ij}^p \le c_{ij}z_{ij}^p \quad \forall (i,j)\in\mathcal{A},\ p=1,\ldots,|\mathcal{P}|, \tag{5.3f}$$

$$\sum_{p=1}^{|\mathcal{P}|} x_{ij}^p \le c_{ij} \quad \forall (i,j)\in\mathcal{A}, \tag{5.3g}$$

$$y_{ij}^p \ge z_{ij}^p - z_{ij}^{p-1} \quad \forall (i,j)\in\mathcal{A},\ p=1,\ldots,|\mathcal{P}|, \tag{5.3h}$$

$$x_{ij}^p,\ y_{ij}^p \ge 0 \quad \forall (i,j)\in\mathcal{A},\ p=1,\ldots,|\mathcal{P}|, \tag{5.3i}$$

$$z_{ij}^p \in \{0,1\} \quad \forall (i,j)\in\mathcal{A},\ p=1,\ldots,|\mathcal{P}|. \tag{5.3j}$$

Objective function (5.3a) minimizes the sum of the activation and transmission costs. Constraints (5.3b) and (5.3c) enforce flow balance conditions. Constraints (5.3d) guarantee that no more than one out-flow arc at s has positive flow on each path $p=1,\ldots,|\mathcal{P}|$, and constraints (5.3e) ensure that positive flows do not split at any intermediate node on each path p. Constraints (5.3f) ensure the flow on each arc (i,j) on each path does not exceed c_{ij}, and these constraints guarantee z_{ij}^p equals 1 if and only if arc (i,j) has positive flow on path p. (This constraint is also valid by the observation that an optimal solution exists in which all flows are integer-valued.) Constraints (5.3g) guarantee that the sum of flow on all paths using each arc $(i,j)\in\mathcal{A}$ does not exceed c_{ij}.

For each arc $(i,j)\in\mathcal{A}$ and path $p=1,\ldots,|\mathcal{P}|$, constraints (5.3h) ensure that y_{ij}^p equals 1 when (i,j) has positive flow on the p^{th} path, but not on the $(p-1)^{st}$ path. Constraints (5.3i) and (5.3j) enforce non-negativity restrictions on variables x and y and binary restrictions on z, respectively. Since z-variables are binary, then $z_{ij}^p - z_{ij}^{p-1}$ either equals -1, 0, or 1. Thus, at optimality, $y_{ij}^p = 0$ when $z_{ij}^p - z_{ij}^{p-1} \le 0$, and $y_{ij}^p = 1$ when $z_{ij}^p - z_{ij}^{p-1} = 1$ for all $(i,j)\in\mathcal{A}$ and $p=1,\ldots,|\mathcal{P}|$. Thus, we

can relax binary restrictions on y_{ij}^p for all $(i,j) \in \mathcal{A}$ and $p = 1, \ldots, |\mathcal{P}|$.

As with the MFP-D, model (5.3) allows for the presence of positive-flow subtours. Such subtours may send flows on cycles within the network, thus avoiding the reactivation of certain arcs. Model (5.3) is a valid formulation for the MCF-A if the given application allows for flow on subtours. For applications that prohibit the existence of subtours, we can dynamically introduce subtour elimination constraints [Laporte, 1986] each time an MCF-A solution contains subtours.

For some optimal solutions, we can shift a sub-sequence of paths in the schedule to obtain an equivalent solution. For example, consider a feasible schedule $\{p_1, p_2, p_3, p_4\}$ having 2, 0, 3, and 5 units of flow, respectively. Since p_2 does not activate any arcs, we could could rearrange paths to obtain another feasible schedule $\{p_1, p_3, p_4, p_2\}$ having 2, 3, 5, and 0 units of flow, respectively. Note that the flow costs of the second solution are equal to the flow costs of the first solution, and the activation costs of the second solution do not exceed those of the first solution. Hence, even if the first solution were optimal, it would be symmetric to the second one.

We thus propose symmetry-breaking constraints that prohibit solutions in which zero-flow paths are interleaved among positive-flow paths. Without the presence of symmetry-breaking constraints, the branch-and-bound (B&B) tree would need to explore many identical solutions of this type; this redundant exploration significantly increases computational time [Sherali and Smith, 2001]. We thus propose the following class of inequalities:

$$\sum_{i:(s,i)\in\mathcal{A}} z_{si}^1 = 1, \tag{5.4a}$$

$$\sum_{i:(s,i)\in\mathcal{A}} z_{si}^{p+1} \leq \sum_{i:(s,i)\in\mathcal{A}} z_{si}^p \quad p = 1, \ldots, |\mathcal{P}| - 1. \tag{5.4b}$$

Constraints (5.4a) ensure that the first path in $\mathcal{P}$ has positive flow, and constraints (5.4b) ensure that the $(p+1)^{st}$ path has positive flow only if the p^{th} path also has positive flow, for all paths $p = 1, \ldots, |\mathcal{P}| - 1$. Observe that (5.4a) and (5.4b) remove the necessity for including constraints (5.3d) in the model. In the given example, adding inequalities (5.4a) and (5.4b) eliminates the equivalent schedule $\{p_1, p_2, p_3, p_4\}$ while maintaining the validity of (5.3).

A classical B&B algorithm can be used to solve model (5.3), but the linear programming relaxation is likely weak since z-variables tend to fractionate when the positive flow on arc (i,j) for a path p is less than c_{ij}. Additionally, the number of positive-flow paths in an optimal solution to (5.3) is likely much fewer than f. Similar to the MFP-D, however, column generation is impractical since

the objective function of the MCF-A relies on the ordering of positive-flow paths. Thus, we propose an iterative relaxation scheme in Section 5.2.3.

5.2.3 Bounding Procedure for the MCF-A

Our bounding procedure is based on a relaxation of the path-based model concept. Our relaxation solutions will consist of a set $\overline{\mathcal{P}}$ of path flows as before, where $|\overline{\mathcal{P}}| < |\mathcal{P}|$, followed by an *aggregate flow* that transmits flows from s to t by any network flow that conserves flow at nodes $\mathcal{V} \setminus \{s,t\}$. Path flows are indexed by $p = 1, \ldots, |\overline{\mathcal{P}}|$ and the aggregate flow is indexed by $|\overline{\mathcal{P}}| + 1$. Note that we must force the aggregate flow to be empty unless the first $|\overline{\mathcal{P}}|$ paths have positive flow, or else all flow in the relaxed solution would be transmitted through the aggregate flow in order to limit activation costs. We call this new relaxation problem MCF-A2.

Variables x, y, and z are defined as before, but for all flows $p = 1, \ldots, |\overline{\mathcal{P}}| + 1$. The MCF-A2 can be modeled by the following MIP.

$$\min \sum_{p=1}^{|\overline{\mathcal{P}}|+1} \sum_{(i,j)\in\mathcal{A}} \left(d_{ij}x_{ij}^p + a_{ij}y_{ij}^p\right) \tag{5.5a}$$

$$\text{s.t.} \sum_{p=1}^{|\overline{\mathcal{P}}|+1} \sum_{i:(s,i)\in\mathcal{A}} x_{si}^p = f, \tag{5.5b}$$

$$\sum_{j:(i,j)\in\mathcal{A}} x_{ij}^p - \sum_{h:(h,i)\in\mathcal{A}} x_{hi}^p = 0 \quad \forall i \in \mathcal{V} \setminus \{s,t\},\ p = 1, \ldots, |\overline{\mathcal{P}}| + 1, \tag{5.5c}$$

$$\sum_{j:(i,j)\in\mathcal{A}} z_{ij}^p - \sum_{h:(hi)\in\mathcal{A}} z_{hi}^p = 0 \quad \forall i \in \mathcal{V} \setminus \{s,t\},\ p = 1, \ldots, |\overline{\mathcal{P}}|, \tag{5.5d}$$

$$z_{ij}^p \le x_{ij}^p \le c_{ij} z_{ij}^p \quad \forall (i,j) \in \mathcal{A},\ p = 1, \ldots, |\overline{\mathcal{P}}| + 1, \tag{5.5e}$$

$$\sum_{p=1}^{|\overline{\mathcal{P}}|+1} x_{ij}^p \le c_{ij} \quad \forall (i,j) \in \mathcal{A}, \tag{5.5f}$$

$$y_{ij}^p \ge z_{ij}^p - z_{ij}^{p-1} \quad \forall (i,j) \in \mathcal{A},\ p = 1, \ldots, |\overline{\mathcal{P}}| + 1, \tag{5.5g}$$

$$\sum_{i:(s,i)\in\mathcal{A}} z_{si}^1 = 1, \tag{5.5h}$$

$$\sum_{i:(s,i)\in\mathcal{A}} z_{si}^{p+1} \le \sum_{i:(s,i)\in\mathcal{A}} z_{si}^p \quad \forall p = 1, \ldots, |\overline{\mathcal{P}}| - 1, \tag{5.5i}$$

$$z_{sj}^{|\overline{\mathcal{P}}|+1} \le \sum_{i:(s,i)\in\mathcal{A}} z_{si}^{|\overline{\mathcal{P}}|} \quad \forall j : (s,j) \in \mathcal{A}, \tag{5.5j}$$

$$\sum_{(u,v)\in\mathcal{A}} y^p_{uv} \geq \sum_{i:(s,i)\in\mathcal{A}} z^p_{si} \quad \forall p = 1,\ldots,|\overline{\mathcal{P}}|+1, \tag{5.5k}$$

$$x^p_{ij},\ y^p_{ij} \geq 0 \quad \forall (i,j)\in\mathcal{A},\ p = 1,\ldots,|\overline{\mathcal{P}}|+1, \tag{5.5l}$$

$$z^p_{ij} \in \{0,1\} \quad \forall (i,j)\in\mathcal{A},\ p = 1,\ldots,|\overline{\mathcal{P}}|+1. \tag{5.5m}$$

Objective function (5.5a) minimizes total flow and activation costs. Constraints (5.5b)–(5.5g) are analogous to constraints (5.3b)–(5.3h), and constraints (5.5h)–(5.5j) are analogous to symmetry-breaking constraints (5.4a) and (5.4b). Constraints (5.5k) ensure that a path cannot transmit flow unless it commits to activating at least one arc on that path. The motivation for using these constraints is that without those constraints, an optimal solution may exist in which paths $1,\ldots,|\overline{\mathcal{P}}|$ are identical, each sending a small flow, in order to avoid activation costs in paths $2,\ldots,|\overline{\mathcal{P}}|$. All remaining flows would then be transmitted in the aggregate flow, further avoiding activation costs and weakening the relaxation. Constraints (5.5k) thus discourage the use of consecutive identical paths, by requiring a new arc activation each time flow is sent across a path (or across the aggregate flow). Finally, constraints (5.5l) and (5.5m) enforce non-negativity restrictions on variables x and y and binary restrictions on z, respectively.

Proposition 4. *Problem MCF-A2 is a relaxation of problem MCF-A.*

Proof. Consider any optimal solution $(\bar{x},\bar{y},\bar{z})$ to MCF-A. We construct a feasible solution $(\hat{x},\hat{y},\hat{z})$ to MCF-A2 whose cost does not exceed that of $(\bar{x},\bar{y},\bar{z})$ in MCF-A. First, we set $\hat{x}^p_{ij} = \bar{x}^p_{ij}$, $\hat{y}^p_{ij} = \bar{y}^p_{ij}$, and $\hat{z}^p_{ij} = \bar{z}^p_{ij}$, $\forall (i,j)\in\mathcal{A}$ and $p = 1,\ldots,|\overline{\mathcal{P}}|$. Then, set $\hat{x}^{|\overline{\mathcal{P}}|+1}_{ij} = \sum_{p=|\overline{\mathcal{P}}|+1}^{|\mathcal{P}|} \bar{x}^p_{ij}$, $\hat{z}^{|\overline{\mathcal{P}}|+1}_{ij} = 1$ if $\hat{x}^{|\overline{\mathcal{P}}|+1}_{ij} > 0$ and $\hat{z}^{|\overline{\mathcal{P}}|+1}_{ij} = 0$ otherwise, and $\hat{y}^{|\overline{\mathcal{P}}|+1}_{ij} = \max\{\hat{z}^{|\overline{\mathcal{P}}|+1}_{ij} - \hat{z}^{|\overline{\mathcal{P}}|}_{ij}, 0\}$, $\forall (i,j)\in\mathcal{A}$. Because $\sum_{p=1}^{|\mathcal{P}|} \bar{x}^p_{ij} = \sum_{p=1}^{|\overline{\mathcal{P}}|+1} \hat{x}^p_{ij}$, $\forall (i,j)\in\mathcal{A}$, and the $\hat{y}$- and $\hat{z}$-values are explicitly defined to satisfy the feasibility criteria, solution $(\hat{x},\hat{y},\hat{z})$ is feasible to MCF-A2.

Next, note that because $\hat{x}^{|\overline{\mathcal{P}}|+1}_{ij} = \sum_{p=|\overline{\mathcal{P}}|+1}^{|\mathcal{P}|} \bar{x}^p_{ij}$, we have that

$$\sum_{p=1}^{|\overline{\mathcal{P}}|+1} \sum_{(i,j)\in\mathcal{A}} d_{ij}\hat{x}^p_{ij} = \sum_{p=1}^{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{A}} d_{ij}\bar{x}^p_{ij}.$$

Also, because for all $(i,j)\in\mathcal{A}$ we have that

- $\hat{z}^{|\overline{\mathcal{P}}|+1}_{ij} = \max_{p\in\{|\overline{\mathcal{P}}|+1,\ldots,|\mathcal{P}|\}}\{\bar{z}^p_{ij}\}$,
- $\bar{y}^p_{ij} = \max\{\bar{z}^p_{ij} - \bar{z}^{p-1}_{ij}, 0\}$, $\forall p = |\overline{\mathcal{P}}|+1,\ldots,|\mathcal{P}|$, at optimality, and
- $\hat{y}^{|\overline{\mathcal{P}}|+1}_{ij} = \max\{\hat{z}^{|\overline{\mathcal{P}}|+1}_{ij} - \hat{z}^{|\overline{\mathcal{P}}|}_{ij}, 0\}$,

we get that $\hat{y}^{|\overline{\mathcal{P}}|+1}_{ij} \leq \sum_{p=|\overline{\mathcal{P}}|+1}^{|\mathcal{P}|} \bar{y}^p_{ij}$. This relationship, along with the fact that $\hat{y}^p_{ij} = \bar{y}^p_{ij}$, $\forall p = 1,\ldots,|\overline{\mathcal{P}}|$ and $(i,j)\in\mathcal{A}$, yields the inequality $\sum_{p=1}^{|\overline{\mathcal{P}}|+1} a_{ij}\hat{y}^p_{ij} \leq \sum_{p=1}^{|\mathcal{P}|} a_{ij}\bar{y}^p_{ij}$. An optimal solution

(x^*, y^*, z^*) to MCF-A2 must have an objective less than or equal to that of $(\hat{x}, \hat{y}, \hat{z})$, whose objective value is less than or equal to that of $(\bar{x}, \bar{y}, \bar{z})$ for MCF-A as demonstrated above. This completes the proof. □

We now describe a bounding procedure that utilizes an optimal MCF-A2 solution to obtain lower and upper bounds for the MCF-A. The lower bound is obtained by solving problem (5.5). To obtain an upper bound, we convert the relaxation solution from MCF-A2 into a feasible solution for MCF-A. To do so, we present the following $\hat{\mathcal{P}}$-decomposition heuristic algorithm. This heuristic pushes flows on the $|\overline{\mathcal{P}}|$ paths given by the solution obtained from (5.5). This solution is optimal if aggregate flow $|\overline{\mathcal{P}}| + 1$ is empty (or if it is a path). Otherwise, our heuristic decomposes the aggregate flow into a schedule of path flows $\hat{\mathcal{P}} = \{\hat{p}_1, \ldots, \hat{p}_{|\hat{\mathcal{P}}|}\}$, with the goal of minimizing the sum of activation costs associated with $\hat{\mathcal{P}}$. For each step i of the $\hat{\mathcal{P}}$-decomposition, define residual flow x_{uv}^{i-1} of each arc $(u, v) \in \mathcal{A}$ to be the aggregate flow on (u, v) minus the sum of flows on (u, v) using the first $(i-1)$ paths in $\hat{\mathcal{P}}$. Let set $\mathcal{A}_{i-1}$ consist of all arcs $(u, v) \in \mathcal{A}$ for which $x_{uv}^{i-1} > 0$ at step i. Letting $\mathcal{A}'_{i-1}$ be the set of arcs in $\mathcal{A}_{i-1}$ used on the $(i-1)^{st}$ path in $\hat{\mathcal{P}}$, our algorithm searches for a path $\hat{p}_i$ that greedily reuses arcs in $\mathcal{A}'_{i-1}$. If an arc $(u, v) \in \mathcal{A}'_{i-1}$ is not included in $\hat{p}_i$, then (u, v) must be activated again on a later path in $\hat{\mathcal{P}}$, and so we search for a path including arcs in $\mathcal{A}'_{i-1}$ having a maximum sum of activation costs.

Given an optimal solution to (5.5), we initialize the $\hat{\mathcal{P}}$-decomposition by letting set $\mathcal{A}_0$ consist of all arcs having positive aggregate flow, and letting set $\mathcal{A}'_0 \subseteq \mathcal{A}_0$ consist of all arcs in $\mathcal{A}_0$ used on path $|\overline{\mathcal{P}}|$. Finally, let the length of each arc $(u, v) \in \mathcal{A}_{i-1}$ be a_{uv} if $(u, v) \in \mathcal{A}'_{i-1}$, and 0 otherwise. Initializing $i = 0$, the $\hat{\mathcal{P}}$-decomposition is as follows.

$\hat{\mathcal{P}}$-decomposition Step 1: If $\mathcal{A}_i = \emptyset$, then terminate with schedule $\hat{\mathcal{P}} = \{\hat{p}_1, \ldots, \hat{p}_i\}$. Otherwise, proceed to $\hat{\mathcal{P}}$-decomposition Step 2.

$\hat{\mathcal{P}}$-decomposition Step 2: If $\mathcal{A}'_i = \emptyset$, then proceed to $\hat{\mathcal{P}}$-decomposition Step 3. Otherwise, set $i = i + 1$, and determine a longest path $\hat{p}_i$ among those arcs in $\mathcal{A}_{i-1}$. Set the flow $f_{\hat{p}_i}$ on $\hat{p}_i$ to be the minimum residual flow among all arcs in $\hat{p}_i$, and decrease the residual flow on each arc in $\hat{p}_i$ by $f_{\hat{p}_i}$. Return to $\hat{\mathcal{P}}$-decomposition Step 1.

$\hat{\mathcal{P}}$-decomposition Step 3: Set $i = i + 1$, and determine a path $\hat{p}_i$ in $\mathcal{A}_{i-1}$ that includes arc $(u, v) \in \text{argmin}_{(j,k)\in\mathcal{A}_{i-1}}\{x_{jk}^{i-1}\}$. Set the flow $f_{\hat{p}_i}$ on $\hat{p}_i$ equal to x_{uv}^{i-1}, and decrease the residual flow

on each arc in $\hat{p}_i$ by $f_{\hat{p}_i}$. Return to $\hat{\mathcal{P}}$-decomposition Step 1.

Letting $\overline{\mathcal{P}} = \{\bar{p}_1, \ldots, \bar{p}_{|\overline{\mathcal{P}}|}\}$, the combined schedule $\{\overline{\mathcal{P}}, \hat{\mathcal{P}}\} = \{\bar{p}_1, \ldots, \bar{p}_{|\overline{\mathcal{P}}|}, \hat{p}_1, \ldots, \hat{p}_i\}$ provides a feasible MCF-A solution and thus corresponds to an upper bound for the MCF-A. Since aggregate flows assume simultaneous transmission, there exists an optimal solution to (5.5) in which no subtours exist in the aggregate flow. Any subtours that do exist can be removed without affecting the objective function value. The resulting graph of aggregate flows is directed and acyclic, and so both the longest path problem in $\hat{\mathcal{P}}$-decomposition Step 2 and the path problem in $\hat{\mathcal{P}}$-decomposition Step 3 can be solved in polynomial time.

The bounding procedure iteratively increases the size of $\overline{\mathcal{P}}$ to improve the lower and upper bounds until an optimal MCF-A solution is found. Letting LB and UB refer to the respective lower and upper bounds on the optimal objective function value for the MCF-A, our bounding procedure is as follows.

Initialization: Solve an instance of the minimum-cost flow problem over arcs in $\mathcal{A}$ to determine flows x, and decompose x into a schedule of paths $\hat{\mathcal{P}}$ using the $\hat{\mathcal{P}}$-decomposition algorithm. Set $|\overline{\mathcal{P}}| = |\hat{\mathcal{P}}|$, $LB = 0$, and UB equal to the total flow and activation cost of schedule $\hat{\mathcal{P}}$. Proceed to the Lower Bound Step.

Lower Bound Step: Solve (5.5), and set LB equal to the optimal objective function value of (5.5). Proceed to the Upper Bound Step.

Upper Bound Step: Execute the $\hat{\mathcal{P}}$-decomposition algorithm to decompose aggregate flows into a schedule $\hat{\mathcal{P}}$. Set UB equal to the minimum between the current value UB value and the total flow and activation cost of the combined schedule $\{\overline{\mathcal{P}}, \hat{\mathcal{P}}\}$. Proceed to the Optimality Check Step.

Optimality Check Step: If $LB = UB$, then terminate with an optimal schedule of flows $\{\overline{\mathcal{P}}, \hat{\mathcal{P}}\}$. Otherwise, increase the size of $\overline{\mathcal{P}}$ by $|\hat{\mathcal{P}}|$, and return to the Lower Bound Step. (If $|\overline{\mathcal{P}}| \geq f$, then exit the bounding procedure and directly solve (5.3).)

Observe that the size of $\overline{\mathcal{P}}$ increases at each non-terminating iteration of the bounding procedure. This observation holds true because $LB < UB$ is only possible when there is positive aggregate flow, resulting in $|\hat{\mathcal{P}}| > 0$ in the Optimality Check Step. Constraints (5.5i) and (5.5j) guarantee that an aggregate flow is empty unless the first $|\overline{\mathcal{P}}|$ paths have positive flow, and constraints

(5.5e) guarantee that the flow, if positive, on each path $\bar{p}_1, \ldots, \bar{p}_{|\overline{\mathcal{P}}|}$ is greater than or equal to 1. Thus, the aggregate flow in an optimal solution to (5.5) is empty and the bounding procedure is guaranteed to converge finitely to an optimal MCF-A solution when $|\overline{\mathcal{P}}| = f$.

We present the following example network in Figure 5.6 to display how the bounding procedure converges. For this example, $f = 11$, and arc labels refer to all c-, d-, and a-values. The

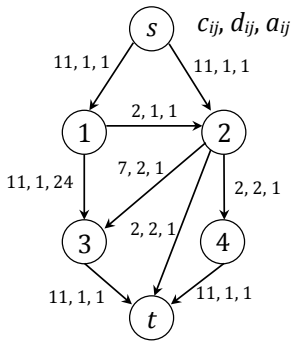


Figure 5.6: Bounding procedure network example

Initialization Step solves an instance of the minimum-cost flow problem that yields a network flow having 11 units of flow on arcs $(s, 1)$, $(1, 3)$, and $(3, t)$. Next, the $\hat{\mathcal{P}}$-decomposition produces a single path $(s, 1, 3, t)$ having 11 units of flow and a minimum cost of 59. Thus, $LB = 0$, $UB = 59$, and $|\overline{\mathcal{P}}| = 1$. Next, the Lower Bound Step solves for an optimal solution to model (5.5) having 7 units of flow on path $\bar{p}_1 = (s, 2, 3, t)$, 4 units of aggregate flow on $(s, 2)$, and 2 units of aggregate flow on arcs $(2, 4)$, $(2, t)$, and $(4, t)$. The total cost of transmitting flow on schedule $\overline{\mathcal{P}}$ is 31, and the total cost of transmitting aggregate flows is 17. Thus, $LB = 48$. The $\hat{\mathcal{P}}$-decomposition algorithm in the Upper Bound Step decomposes the aggregate flow into a schedule $\hat{\mathcal{P}}$ containing paths $\hat{p}_1 = (s, 2, t)$ and $\hat{p}_1 = (s, 2, 4, t)$ each having 2 units of flow. The cost of transmitting flow on schedule $\hat{\mathcal{P}}$ is 17, and the $UB = 48$. Since $LB = UB$, the bounding procedure terminates in the Optimality Check Step with an optimal schedule of path flows $\{\overline{\mathcal{P}}, \hat{\mathcal{P}}\}$ having a minimum cost of 48.

5.2.4 Bounding Procedure Computational Results

In this section we compare the performance of our bounding procedure with the MIP (5.3). We solve all mathematical programming models using CPLEX 12.8 via ILOG Concert Technology, and we performed all experiments on a computer having a 2.9GHz Dual-core Intel i7 processor with 8GB RAM. We present all CPU times in seconds and impose a 3600 second time limit. For

all instances, we consider randomly-generated integer c-, d-, and a-values uniformly distributed between $[1, 10]$, $[1, 15]$, and $[1, 4]$, respectively, for dense networks having an arc between every distinct node pair except between s and t. We first generate five random instances of the MCF-A for each combination of networks having 10, 12, and 15 relay nodes and flow $f \in \{10, 11, 12, 13, 14, 15\}$.

Tables 5.3, 5.4, and 5.5 display the computational results of both approaches for networks having 10, 12, and 15 nodes, respectively. Each table depicts the average CPU time, average size of $|\overline{\mathcal{P}}|$ at optimality, the total number of instances solved given an optimality gap tolerance of 0.1%, and the average optimality gap at termination. For those instances that are terminated due to the time limit, we record the current optimality gap observed after 3600 seconds, and count the CPU time required for the instance as 3600 seconds.

Table 5.3: Average MCF-A results over five networks having 10 nodes

	CPU Time (sec.)		Average Size of $\lvert\overline{\mathcal{P}}\rvert$		# Solved (out of 5)		Optimality Gap	
f	MIP	Bounding	MIP	Bounding	MIP	Bounding	MIP	Bounding
10	6.46	0.32	–	2.8	5	5	0%	0%
11	115.21	10.71	–	4.4	5	5	0%	0%
12	123.53	22.84	–	5.6	5	5	0%	0%
13	183.87	5.71	–	4.5	5	5	0%	0%
14	237.15	10.81	–	5.4	5	5	0%	0%
15	835.80	88.26	–	4.8	5	5	0%	0%

Table 5.3 displays the computational results for networks having 10 nodes. In all instances, both techniques solved the MCF-A to optimality within the given time limit. Additionally, our bounding procedure required less CPU time than the MIP. For example, when $f = 10$, our procedure solved all instances in less than one second while the MIP required at least five seconds. As flow f increased, the CPU time for solving the MIP also increased. Alternatively, our bounding procedure was less dependent on the value of f. Our procedure seemed to be more influenced by the size of $|\overline{\mathcal{P}}|$. For example, our procedure required less time, on average, to solve the MCF-A for $f = 13$ than for $f = 12$. The maximum CPU time required by our bounding procedure to solve any instance was 421 seconds for $f = 15$ and $|\overline{\mathcal{P}}| = 6$, at optimality. Alternatively, an instance for the same value of f required less than one second to solve, but yielded $|\overline{\mathcal{P}}| = 3$ at optimality. Therefore, we observe our procedure performs well when a small number of positive-flow paths are needed to transmit f units of flow from s to t.

Table 5.4 next displays the computational results for networks having 12 nodes. As before,

Table 5.4: Average MCF-A results over five networks having 12 nodes

	CPU Time (sec.)		Average Size of $\lvert\overline{\mathcal{P}}\rvert$		# Solved (out of 5)		Optimality Gap	
f	MIP	Bounding	MIP	Bounding	MIP	Bounding	MIP	Bounding
10	18.07	4.78	–	4.6	5	5	0%	0%
11	333.98	34.89	–	5	5	5	0%	0%
12	283.91	20.95	–	4.4	5	5	0%	0%
13	401.56	6.65	–	4.2	5	5	0%	0%
14	711.40	113.34	–	5.6	5	5	0%	0%
15	1521.13	331.10	–	6.6	4	5	0.04%	0%

the average CPU time for the MIP increased as the size of f increased. While the MIP solved all previous instances to optimality, it failed to solve one instance for $f = 15$. Unsurprisingly, the average CPU time for solving the MIP increased as the size of the network increased from 10 nodes to 12 nodes. Our bounding procedure, on average, required more time to solve the MCF-A over networks having 12 nodes when compared to 10 nodes. Alternatively, some instances required less time when $|\overline{\mathcal{P}}|$ was smaller at optimality.

Table 5.5: Average MCF-A results over five networks having 15 nodes

	CPU Time (sec.)		Size of $\lvert\overline{\mathcal{P}}\rvert$		# Solved (out of 5)		Opt. Gap	
f	MIP	Bounding	MIP	Bounding	MIP	Bounding	MIP	Bounding
10	43.95	4.86	–	3.6	5	5	0%	0%
11	669.39	67.31	–	4.6	5	5	0%	0%
12	200.82	10.05	–	4.2	5	5	0%	0%
13	157.14	8.13	–	4	5	5	0%	0%
14	864.63	98.77	–	5.2	5	5	0%	0%
15	499.77	32.48	–	4.4	5	5	0%	0%

Table 5.5 displays the computational results for networks having 15 nodes. Our bounding procedure continued to perform well when compared to the MIP. Much like networks having 10 or 12 nodes, the average CPU time of our bounding procedure increased as the average number of paths in $\overline{\mathcal{P}}$ increased. As observed in both tables, the size of the network had a relatively small effect on the average CPU time of our bounding procedure. The bounding procedure did not require significantly more time to solve the MCF-A over networks having 15 nodes than it did for 10 or 12 nodes.

Our procedure performs well when a small number of positive-flow paths are needed to transmit f units of flow from s to t. Alternatively, we test the scalability of our bounding procedure

to determine for what types of networks our procedure does not perform as well. We execute the bounding procedure to solve five randomly-generated MCF-A instances over networks having 20 nodes and values $f \in \{15, 20, 25\}$. The MIP reached the time limit in all instances for networks of this size and flow value.

Table 5.6: Average bounding procedure results over five networks having 20 nodes

f	CPU Time (sec.)	Average Size of $\lvert\overline{\mathcal{P}}\rvert$	# Solved (out of 5)	Optimality Gap
15	144.79	6.2	5	0%
20	864.63	6	4	7.62%
25	3459.49	10.2	1	6.59%

Table 5.6 displays that, on average, the CPU time of the bounding procedure increases as f increases for networks having 20 nodes. Additionally, we observe that, as f increases, our bounding procedure is less likely to be able to solve the MCF-A since more number of positive-flow paths are needed for larger values of f. For example, bounding procedure was only able to solve one instance for $f = 25$, and it still required around 2300 seconds to solve this instance. When the bounding procedure was not able to solve the MCF-A, the average optimality gap was 7.6% and 6.6% for $f = 20$ and $f = 25$, respectively. These results illustrate the limitations of our bounding procedure as it relates to network characteristics.

Chapter 6

Conclusions

In this dissertation, we presented models and algorithms for solving various classes of network flow problems that arise in WSN applications. Spurred by the growing interest in WSN optimization, Chapter 2 first surveyed various research approaches and extensions to the WSN lifetime maximization problem that include online routing, clustering approaches, and lifetime maximization on specially structured networks. We also considered the impact of having mobile and/or multiple sinks and delay-tolerant routing, and we detailed future areas of research.

In Chapter 3 we studied a variation of the maximum flow problem having a set of node-capacity restrictions. We highlighted the case in which each node $i \in \mathcal{V}$ has a finite capacity $b_i > 0$, and a unit of flow on arc $(i, j) \in \mathcal{A}$ consumes $g_{ij} > 0$ units of capacity at i. We presented two augmenting-flow algorithms as alternatives to solving a large LP. The first algorithm is an almost-exact solution technique that iteratively solves two auxiliary LP models. These LPs either terminate the algorithm with an optimal or close to optimal solution to the NCMFP or augment flow on a set of cycles to increase node capacities and then on an s–t path to maximize additional flow in the network. For situations that necessitate the complete avoidance of linear programming techniques, our second approach is a heuristic algorithm that alters the cycle identification portion of the first approach.

The almost-exact approach required less time to solve the NCMFP when compared to solving a single LP for the NCMFP. The savings in computational time was significant for large networks, and this approach is especially useful when a network contains a relatively small set of particular nodes that are susceptible to exhaustion. Our heuristic approach is an adequate option for solving the NCMFP when linear programming solvers are unavailable.

Some WSN applications consider the presence of in-flow costs $h_{ij} > 0$ incurred at each node $j \in \mathcal{V}$ when arc (i, j) has positive flow. Both algorithms presented in Chapter 3 could be amended to consider the presence of such costs, but the heuristic approach could become time-consuming and tedious. Thus, we encourage future studies to examine the NCMFP having such costs. Future studies may also investigate solution approaches to solving the NCMFP when network information (e.g., node capacity) may be unknown or uncertain.

Next, Chapter 4 introduced a variation of the maximum flow problem having semicontinuous restrictions (MFP-S) on path flows. Since an MIP formulation for the MFP-S is pseudo-polynomial in size and difficult to solve, we proposed a B&P algorithm along with a specialized branching procedure based on the presence of a cut-set in a non-feasible solution. Our computational results show that the B&P approach significantly reduces the computational time required to solve the MFP-S to optimality.

Extensions of this work could include exploring other variations on semicontinuous flow restrictions. For example, an interesting research direction may be to investigate multi-modal shipping problems, in which the lower bound on positive path flows depends on the mode of transportation. These problems may also incorporate costs of aggregating flows at nodes, such as would be required when moving from one mode of transportation to the next. Whereas our work explicitly enforces semicontinuity, other problems could relax the semicontinuity assumption by penalizing via a fixed cost each instance in which positive flows are less than some lower bound.

Finally, Chapter 5 considered network applications that require dynamic flows to be transmitted according to a non-simultaneous schedule. We first considered a network flow problem, in which an arc is required to transmit a minimum amount of flow each time it begins transmitting flow. We modeled this problem as a maximum flow problem having dynamic semicontinuous restrictions (MFP-D) on path flows. An MIP for solving this problem typically requires significant amount of time to solve, and will ultimately require the use of heuristics in many practical settings. Thus, we provided heuristic algorithms to obtain lower and upper bounds for the problem. We leave the development of advanced MFP-D heuristics for future study.

We next considered the presence of setup costs in dynamic network flow applications, in which a fixed cost is incurred each time an arc begins transmission. This problem can be modeled by the minimum-cost flow problem having *arc-activation* costs (MCF-A), in which an arc (i, j) is said to be *activated* on path p when (i, j) has positive flow on the p^{th} scheduled path, but not on the

$(p-1)^{st}$ scheduled path. As an alternative to solving an MIP, we introduced an exact algorithm for computing an optimal MCF-A solution. In all instances, this approach provided an improvement in computational time over the MIP. As the amount of flow to be transmitted from source to sink increases or the size of the network grows, our algorithm significantly decreases the time required to solve the MCF-A when compared to the MIP.

Future research needs also to examine other problems in non-simultaneous flows, such as multi-stage network flow problems, and in problems having stochastic parameters. For example, consider the multi-period investment problem having uncertain returns and minimum investment levels for single- or multi-period investments. In this case, investment decisions must be made at the beginning of each period, and so flows, or investments, along arcs may have some minimum investment level. An optimization algorithm would need to anticipate the possibility of having smaller-than-anticipated returns in the investment plan, hedging against the possibility that insufficient returns may be present in a period for which an investment was planned.

Appendices

Appendix A

How to determine upper concave hull $\mathcal{C}_k^{UB}$

Let $\mathcal{C}_k$ be the set of constraints (3.11a), (3.11c), and (3.11d) for cycle $\theta_k \in \Theta$;
Identify constraint $c \in \mathcal{C}_k$ having maximum slope;
Remove c from $\mathcal{C}_k$ and add c to $\mathcal{C}_k^{UB}$;
while $\mathcal{C}_k \neq \emptyset$ **do**
  Identify constraint $c \in \mathcal{C}_k$ having maximum slope;
  Identify constraint $c' \in \mathcal{C}_k^{UB}$ having minimum slope;
  Identify constraint $c'' \in \mathcal{C}_k^{UB}$ having the second minimum slope;
  Determine the intersection point $e_{cc'}$ of constraints c and c';
  if $e_{cc'}$ *is feasible to* c'' *or no* c'' *exists* **then**
    Remove c from $\mathcal{C}_k$ and add c to $\mathcal{C}_k^{UB}$;
  else
    Remove c' from $\mathcal{C}_k^{UB}$;
  end
end

Algorithm 1: Algorithm to determine upper concave hull $\mathcal{C}_k^{UB}$

A similar algorithm can be used to determine the lower convex hull $\mathcal{C}_k^{LB}$.

Modified augmenting-flow (m-AF) algorithm

Let $\mathcal{A}'$ represent the set of residual arcs corresponding to flows $\bar{x}$. For all forward arcs $(i,j) \in \mathcal{A}' \cap \mathcal{A}$, define $\bar{c}_{ij} = c_{ij} - \bar{x}_{ij}$ as the residual capacity on (i,j). For all reverse arcs $(i,j) \in \mathcal{A}' \setminus \mathcal{A}$, define $\bar{c}_{ij} = \bar{x}_{ji}$ as the residual capacity on (i,j). Define $\bar{b}_i = b_i - \sum_{j:(i,j)\in\mathcal{A}} g_{ij}\bar{x}_{ij}$ to be the residual capacity at each node $i \in \mathcal{V}$. Thus, define $\hat{c}_{ij} = \min\{\bar{c}_{ij},\ \bar{b}_i/g_{ij}\}$ as the maximum allowable flow on each forward arc $(i,j) \in \mathcal{A}' \cap \mathcal{A}$. Additionally, define $\hat{c}_{ij} = \bar{c}_{ij}$ as the maximum allowable flow on each reverse arc $(i,j) \in \mathcal{A}' \setminus \mathcal{A}$. Finally, let $\mathcal{V}_R$ contain all nodes $i \in \mathcal{V}$ for which $\bar{b}_i = 0$.

m-AF Initialization: For all $(i,j) \in \mathcal{A}'$, set $\bar{x}_{ij} = 0$ and $\bar{c}_{ij} = c_{ij}$. Set $\bar{b}_i = b_i,\ \forall i \in \mathcal{V}$, and $\hat{c}_{ij} = \min\{\bar{c}_{ij},\ \bar{b}_i/g_{ij}\},\ \forall (i,j) \in \mathcal{A}$. Set $\mathcal{V}_R = \emptyset$ (since $b_i > 0,\ \forall i \in \mathcal{V}$).

m-AF Step 1: Execute a DFS originating at s to determine a s–t path p containing only arcs $(i,j) \in \mathcal{A}'$ for which $\hat{c}_{ij} > 0$. If no such path exists, proceed to m-AF Step 3. Otherwise, let $\mathcal{A}_p$ be the set of all arcs $(i,j) \in \mathcal{A}'$ contained within path p, and let $\mathcal{V}_p$ be the set of all nodes $i \in \mathcal{V}$ visited by p. Next, let $\delta = \min\{\hat{c}_{ij} : (i,j) \in \mathcal{A}_p\}$. For each arc $(i,j) \in \mathcal{A}_p$, augment δ units of flow on each forward arc $(i,j) \in \mathcal{A}' \cap \mathcal{A}$ by setting $\bar{x}_{ij} = \bar{x}_{ij} + \delta$ and on each reverse arc $(j,i) \in \mathcal{A}' \setminus \mathcal{A}$ by setting $\bar{x}_{ij} = \bar{x}_{ij} - \delta$. Finally, for each arc $(i,j) \in \mathcal{A}_p$, update residual capacities $\bar{c}_{ij}$ and $\bar{c}_{ji}$ for arcs $(i,j),\ (j,i) \in \mathcal{A}'$, respectively. If $(i,j) \in \mathcal{A}' \cap \mathcal{A}$, then update $\bar{b}_i$ for node $i \in \mathcal{V}_p$, and if $(i,j) \in \mathcal{A}' \setminus \mathcal{A}$, then update $\bar{b}_j$ for node $j \in \mathcal{V}_p$. Proceed to m-AF Step 2.

m-AF Step 2: For all forward arcs $(i,j) \in \mathcal{A}' \cap \mathcal{A} \cap \mathcal{A}_p$, set the maximum allowable flow value on $(i,j) \in \mathcal{A}'$ to be $\hat{c}_{ij} = \min\{\bar{c}_{ij}, \bar{b}_i / g_{ij}\}$. For all reverse arcs $(i,j) \in \{\mathcal{A}' \setminus \mathcal{A}\} \cap \mathcal{A}_p$, set $\hat{c}_{ij} = \bar{c}_{ij}$. Return to m-AF Step 1.

m-AF Step 3: The m-AF terminates with flows $\bar{x}$ feasible to model (3.1) for the NCMFP having a maximum flow value $f = \sum_{i:(s,i)\in\mathcal{A}} \bar{x}_{si}$. The balance of flow is preserved and node- and arc-capacity constraints are satisfied at each step of the m-AF since the m-AF augments a common amount of flow along an entire s–t path with each augmentation.

Appendix B

Proof of Theorem 2

Proof. Our transformation is from the 3-Partition (3PART) problem [Garey and Johnson, 1975] defined as follows.

> INSTANCE: Set $N = \{n_1, n_2, \ldots, n_k\}$ of $k = 3m$ elements of size $a(n_i) \in \mathbb{Z}^+$, a bound $\tau \in \mathbb{Z}^+$, such that $\tau/4 < a(n_i) < \tau/2$, for each $n_i \in N$, and $\sum_{i=1}^{k} a(n_i) = m\tau$.
>
> QUESTION: Can N be partitioned into m disjoint sets $N_1, N_2, \ldots, N_m$ such that, for each $i = 1, \ldots, m$, $\sum_{n_j \in N_i} a(n_j) = \tau$ (so that each N_i must contain exactly three elements from N)?

We transform an instance of 3PART into an SPDP instance as follows. Let $\mathcal{V}$ consist of nodes $\{s, 1, \ldots, k+m+1, t\}$, and define $\mathcal{V}_1 = \{1, \ldots, k\}$ and $\mathcal{V}_2 = \{k+2, \ldots, k+m+1\}$. Let the aggregate flow on arcs (s, i) and $(i, k+1)$ be $a(n_i)$, $\forall i \in \mathcal{V}_1$. Additionally, let the aggregate flow on arcs $(k+1, m)$ and (m, t) be τ, for each $m \in \mathcal{V}_2$. Set $\ell = \tau/4$.

To show equivalence of these problems, first suppose that the 3PART instance is a yes-instance. We construct an SPDP solution as follows. For each $v = 1, \ldots, m$, examine the set $N_v = \{n_h, n_i, n_j\}$. For element n_h, we create an SPDP path $p_h = (s, h, k+1, k+1+v, t)$ having a flow of $a(n_h)$ units. The aggregate flow on arcs (s, h) and $(h, k+1)$ are satisfied by path p_h, and since $a(n_h) > \tau/4$, the flow on p_h is stable. Repeat this step for n_i and n_j as well, creating paths p_i and p_j. The sum of flows on arcs $(k+1, k+v+1)$ and $(k+v+1, t)$ is given by $a(n_h) + a(n_i) + a(n_j) = \tau$. Repeating this construction for all $v = 1, \ldots, m$ yields a decomposition of stable paths from the given stable aggregate arc flows. Thus, SPDP has a solution.

Conversely, suppose there exists an SPDP solution. For this solution, suppose first that there exist two paths p'_h and p''_h originating on arc (s, h), $h \in \mathcal{V}_1$ that have positive flows. If the flow on p'_h is $\geq t/4$ (but $< a(n_h)$), then the resulting flow on p''_h must be strictly less than $\tau/4$ since $a(n_h) < \tau/2$. Therefore, the aggregate flow on each arc (s, h) (and $(h, k+1)$) must be satisfied by a single path p_h having a flow of $a(n_h)$. Additionally, there exists some set N_v of stable positive-flow paths whose flows satisfy the aggregate flow τ on arc $(k+1, k+v+1)$ and $(k+v+1, t)$, for $v = 1, \ldots, m$. Since $\tau/4 < a(n_i) < \tau/2$, $\forall i = 1, \ldots, t$, each N_v corresponds to exactly three disjoint paths $\{p_h, p_i, p_j\}$ having flow values of $a(n_h)$, $a(n_i)$, and $a(n_j)$, respectively, with $a(n_h) + a(n_i) + a(n_j) = \tau$. Letting $N_v = \{n_h, n_i, n_j\}$ for each v, we recover a 3PART solution.

We have shown that 3-PART $\leq_p$ SPDP; therefore, SPDP is NP-hard. □

Appendix C

An example showing that a given arc-stable solution is not also dynamic stable

Let an *arc-stable solution* to the MFP problem be one in which the sum of positive flows along each arc among all paths is greater than or equal to the lower bound ℓ. Thus, an arc stable solution satisfies the following constraint $\sum_{p \in S} x_{ij}^p \in \{0, [\ell, u]\}$ To show that an arc-stable solution is not necessarily dynamic stable, we examine the process of constructing a dynamic-stable solution from a given arc-stable solution. In the context of constructing a solution whose aggregate flows match a given arc-stable solution, we first introduce the notion of an *active* arc. Consider an ordered set of paths $p_1, \ldots, p_v$ with corresponding flows $f_1, \ldots, f_v$. An active arc is one that must transmit positive flow on path $v+1$ in any dynamic-stable solution including paths $p_1, \ldots, p_v$ and corresponding flows $f_1, \ldots, f_v$. An active arc (i, j) is deemed either *active-nonstable* or *active-stable*. Suppose that arc (i, j) belongs to all paths $p_k, p_{k+1}, \ldots, p_v$ but not p_{k-1}. Arc (i, j) is active-nonstable if $\sum_{h=k}^{v} f_h < \ell$. An arc (i, j) is active-stable if $\sum_{h=k}^{v} f_h \geq \ell$, but the remaining aggregate flow that must be sent on arc (i, j) is less than ℓ. Consider the example in Figure 1, where $\ell = 10$. We show that there does not exist a dynamic-stable solution, even though the aggregate arc flows are stable.

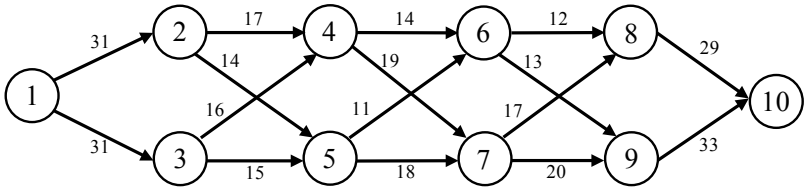


Figure 1: Arc-stable solution with $\ell = 10$

Suppose there exists a schedule of paths corresponding to the aggregate arc-stable solution in Figure 1 that meets dynamic-stable restrictions. This schedule may begin with one of the sixteen unique paths from node 1 to node 10. Consider the case in which the schedule begins with path $p_1 = (1, 2, 5, 6, 9, 10)$. The flow on p_1 cannot exceed 11, which is the minimum flow among all arcs in p_1. If the flow on p_1 were $\geq \ell$ and < 11, then the next path must include all arcs in $(2, 5)$, $(5, 6)$, and $(6, 9)$, or else no subsequent set of paths transmitting the flows on those arcs would be able to send enough consecutive flow over those arcs to meet the stability restrictions. However, p_1 is the only path including those arcs, and so no unique path could follow p_2 in this case. For the same reason, no flow on the first path can be less than ℓ, or else the next path must include all arcs in p_1. Therefore,

the flow on p_1 must be exactly 11. After p_1 sends 11 units of flow, Figure 2 displays the remaining flow. Arcs $(2, 5)$ and $(6, 9)$ have 3 and 2 remaining units of flow to send, respectively. These arcs are active-stable because the flow on the first path p_1 is greater than or equal to ℓ. No paths exist in the network in Figure 2 that contain both of these arcs; therefore, a dynamic-stable solution cannot begin with p_1. Now, consider when the schedule begins with path $\bar{p}_1 = (1, 2, 4, 6, 8, 10)$. As before,

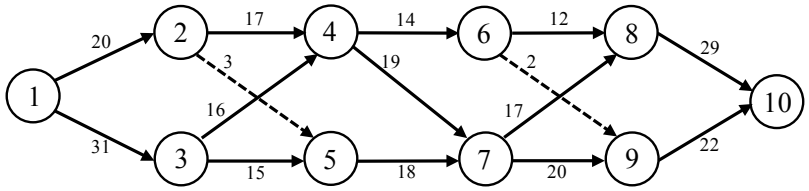


Figure 2: Updated network after p_1

this path must transmit 12 units of flow. Arcs $(2, 4)$ and $(4, 6)$ become active-stable in the updated network (Figure 3) because they have 5 and 2 units of remaining flow, respectively. The next path

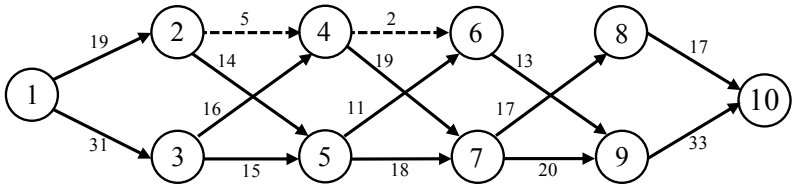


Figure 3: Updated network after $\bar{p}_1$

in the schedule must then be $\bar{p}_2 = (1, 2, 4, 6, 9, 10)$, having 2 units of flow. After $\bar{p}_2$ sends flow, arc $(2, 4)$ remains active-stable and arcs $(6, 9)$ and $(9, 10)$ become active-nonstable (Figure 4). No paths exist in the updated network having flow on both arcs $(2, 4)$ and $(6, 9)$; therefore, a schedule of paths meeting dynamic-stable restrictions cannot begin with $\bar{p}_1$. Similar analysis (omitted for brevity) yields the same result for the 14 other possible initial paths. As a result, no dynamic-stable solution corresponds to the flows in Figure 1, and we conclude that an arc-stable solution is not necessarily dynamic stable.

Finding paths in Steps 2a and 2b of the MFP-D heuristic

For Step 2a, we identify (using breadth-first search (BFS)) a path from s to any node i such that $r_{ij}^k = r_{min}^k$, using only arcs having positive residual values (i.e., $(i, j) \in \mathcal{A} : r_{ij}^k > 0$). We then repeat this search for a path from j to t using only arcs having positive residual flow. We return a

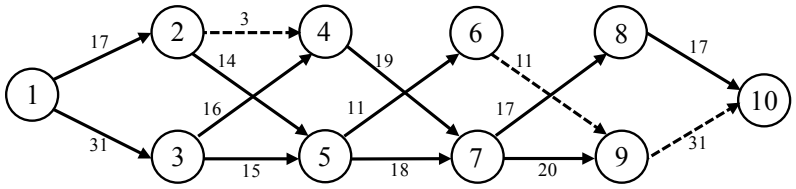


Figure 4: Updated network after $\bar{p}_2$

path p_k that joins subpaths s–i and j–t with arc (i, j).

For Step 2b, we join a series of "subpaths" as in the Step 2a strategy. Each subpath is identified using only positive residual-flow arcs, and each is computed using BFS. We first establish a topological ordering (see, e.g., [Ahuja et al., 1993]) of nodes in the network of positive residual-flow arcs $(i, j) \in \mathcal{A} : r_{ij}^k > 0$. This ordering exists by assumption that the input arc-stable aggregate flows are acyclic. Assuming that the nodes are now reindexed in topological order, we then create a subpath from s to i' such that $i' = \min\{i : (i, j) \in \mathcal{A}^k\}$ and join subpath s–i' with active arc $(i', j') \in \mathcal{A}^k$. Next, we search for a subpath from j' to node u', such that $u' = \min\{u : (u, v) \in \mathcal{A}^k : u > j\}$ and extend our previous path with the subpath from j'–u', followed by arc $(u', v') \in \mathcal{A}^k$. This process continues until a path from s–v is established, for some v, that uses all active arcs. We then find one more subpath from v to t, join path s–v with subpath v–t, and set p_k equal to the s–t path. If at any point a subpath cannot be identified, then a path sought at Step 2b is not found.

Finding paths in Improvement Step 2 of the MFP-D heuristic

For Improvement Step 2, we also join a series of subpaths to create a path $\bar{p}_{k+1}$ containing all active non-stable arcs in $\mathcal{A}^{k+1}$. Our goal is to maximize flow on $\bar{p}_{k+1}$ in a heuristic manner without violating any arc- or node-capacity constraints. To construct $\bar{p}_{k+1}$, we first define a_{ij} to be the maximum possible flow on arc (i, j) without violating node- or arc-capacity constraints, given prior flows transmitted in our heuristic solution. Letting $\bar{b}_i$ and $\bar{c}_{ij}$ be the remaining capacities on node i and arc (i, j), respectively, after the flows on paths $p_1, \ldots, p_k$ are sent, then the allowable flow on (i, j) is $a_{ij} = \min\{\bar{b}_i/g_{ij}, \bar{c}_{ij}\}$. Additionally, we define $f^{h,j}$ as the maximum allowable flow on a path from node h to node j without violating any node or arc capacity constraints along the h–j path.

Letting $\bar{\mathcal{A}}^{k+1}$ be the set of active non-stable arcs, we first search for a *widest* path from s to the tail node i of each arc $(i, j) \in \bar{\mathcal{A}}^{k+1}$. A widest path is one that maximizes the minimum

flow that can be sent on the path. To find these widest paths, we implement a simple adaptation of Dijkstra's algorithm to solve the bottleneck path problem [Punnen, 1991]. Since all a-values are non-negative, this search will always a set of simple s–i paths. Thus, the maximum allowable flow $f^{s,i}$ on each s–i path is equal to the minimum a-value among all arcs in the path from s to i. We select an s–i path having the maximum $f^{s,i}$-value, join this path with (i,j), remove (i,j) from $\bar{\mathcal{A}}^{k+1}$, and set $f^{s,j} = \min\{f^{s,i}, a_{ij}\}$.

We then search for a widest path from j to the tail u of each arc $(u,v) \in \bar{\mathcal{A}}^{k+1}$. From among these simple paths, we select one having a maximum $f^{j,u}$-value, join s–j, j–u, and arc (u,v), and remove (u,v) from $\bar{\mathcal{A}}^{k+1}$. The resulting s–v path may consist of subtours since some node may be contained in both s–j and j–u. Thus, we define $T^{s,v}$ to be the set of nodes visited multiple times in s–v, $\mathcal{A}_{h,s,v}$ to be the set of arcs leaving node $h \in T^{s,v}$, and set the flow as:

$$f^{s,v} = \min\left\{ f^{s,j}, f^{j,v}, a_{uv}, \min\left\{ \frac{\bar{b}_h}{\sum_{(h,k)\in\mathcal{A}_{h,s,v}} g_{hk}} : h \in T^{s,v} \right\} \right\}.$$

We repeat our search in this fashion: We set node j to refer to node v, and reiterate by once again looking for a widest path j–u containing an arc $(u,v) \in \bar{\mathcal{A}}^{k+1}$, until either no such path exists or $\bar{\mathcal{A}}^{k+1}$ is empty. In the former case, we are unable to find a path $\bar{p}_{k+1}$. In the latter case, we identify a widest path from v to t (known to exist by assumption), join s–v and v–t, set $\bar{p}_{k+1}$ equal to path s–t having flow:

$$\bar{f}_{k+1} = \min\left\{ f^{s,v}, f^{v,t}, \min\left\{ \frac{\bar{b}_h}{\sum_{(h,k)\in\mathcal{A}_{h,s,t}} g_{hk}} : h \in T^{s,t} \right\} \right\}.$$

Finding paths in Improvement Step 4 of the MFP-D heuristic

This procedure is similar to Improvement Step 2, but without the complicating requirement of including active non-stable arcs. We use a variation of Dijkstra's algorithm for the bottleneck path problem to identify a path $\bar{p}_{k+h}$ that maximizes the flow from s to t without violating any node or arc capacity constraints. Note that there will always exist an optimal solution to this problem that is a simple path, unlike the case of Improvement Step 2. If the allowable flow along this path is less than ℓ, then we are unable to determine a static-stable path $\bar{p}_{k+h}$.

Bibliography

A. A. Abbasi and M. Younis. A survey on clustering algorithms for wireless sensor networks. *Computer Communications*, 30(14):2826–2841, 2007.

M. M. Afsar and M. Tayarani-N. Clustering in sensor networks: A literature survey. *Journal of Network and Computer Applications*, 46:198–226, 2014.

C. C. Aggarwal and J. B. Orlin. On multiroute maximum flows in networks. *Networks*, 39(1):43–52, 2002.

R. K. Ahuja and J. B. Orlin. A fast and simple algorithm for the maximum flow problem. *Operations Research*, 37(5):748–759, 1989.

R. K. Ahuja and J. B. Orlin. Equivalence of the primal and dual simplex algorithms for the maximum flow problem. *Operations Research Letters*, 20:101–108, 1997.

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Some recent advances in network flows. *SIAM Review*, 33(2):175–219, 1991.

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

K. Akkaya and M. Younis. A survey on routing protocols for wireless sensor networks. *Ad hoc networks*, 3(3):325–349, 2005.

A. Alfieri, A. Bianco, P. Brandimarte, and C. F. Chiasserini. Maximizing system lifetime in wireless sensor networks. *European Journal of Operational Research*, 181(1):390–402, 2007.

H. M. Ammari. On the energy-delay trade-off in geographic forwarding in always-on wireless sensor networks: A multi-objective optimization problem. *Computer Networks*, 9(19):1913–1935, 2013.

G. Anastasi, M. Conti, M. D. Francesco, and A. Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537–568, 2009.

G. Angulo, S. Ahmed, and S. Dey. Semi-continuous network flow problems. *Mathematical Programming*, 145(1–2):565–599, 2014.

R. D. Armstrong, W. Chen, D. Goldfarb, and Z. Jin. Strongly polynomial dual simplex methods for the maximum flow problem. *Mathematical Programming*, 80(1):17–33, 1998.

J. Aslam, Q. Li, and D. Rus. Three power-aware routing algorithms for sensor networks. *Wireless Communications and Mobile Computing*, 3:187–208, 2003.

A. A. Assad. Multicommodity network flows–a survey. *Networks*, 8(1):37–91, 1978.

C. Barnhart, C. A. Hane, E. L. Johnson, and G. Sigismondi. A column generation and partitioning approach for multi-commodity flow problems. *Telecommunication Systems*, 3(3):239–258, 1995.

G. Barrenetxea, F. Ingelrest, G. Schaefer, M. Vetterli, O. Couach, and M. Parlange. SensorScope: Out-of-the-box environmental monitoring. In *Proceedings of the 7th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 332–343, St. Louis, MO, 2008. ACM/IEEE.

S. Basagni, C. Carosi, and C. Phillips. Moving multiple sinks through wireless sensor networks for lifetime maximization. In *Proceedings of 5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems MASS*, Atlanta, GA, 2008. IEEE.

S. Basagni, A. Carosi, and C. Petrioli. Heuristics for lifetime maximization in wireless sensor networks with multiple mobile sinks. In *Proceedings of IEEE International Conference on Communications*, pages 1–6, Dresden, Germany, 2009. IEEE.

E. Beale. Branch-and-bound methods for numerical optimization. In M. M. Barritt and D. Wishart, editors, *COMPSTAT 80: Proceedings in Computational Statistics*, pages 11–20, Edinburgh, 1980. Physica Verlag.

E. Beale. Integer programming. *Computational Mathematical Programming*, NATO ASI Series(f15): 1–24, 1985.

E. M. Beale. Branch and bound methods for mathematical programming systems. *Annals of Discrete Mathematics*, 5:201–219, 1979.

B. Behdani, Y. S. Yun, J. C. Smith, and Y. Xia. Decomposition algorithms for maximizing the lifetime of wireless sensor networks with mobile sinks. *Computers & Operations Research*, 39(5): 1054–1061, 2012.

B. Behdani, J. C. Smith, and Y. Xia. The lifetime maximization problem in wireless sensor networks with a mobile sink: MIP formulations and algorithms. *IIE Transactions*, 45(10):1094–1113, 2013.

A. Ben-Tal and A. Nemirovski. Robust optimization–methodology and applications. *Mathematical Programming*, 92(3):453–480, 2002.

D. Bienstock. Computational study of a family of mixed-integer quadratic programming problems. *Mathematical Programming*, 74:121–140, 1996.

H. L. Bodlaender, R. B. Tan, T. C. van Dijk, and J. van Leeuwen. Integer maximum flow in wireless sensor networks with energy constraint. In J. Gudmundsson, editor, *Proceedings of the 11th Scandinavian Workshop on Algorithm Theory*, 5124, pages 102–113, Gottenburg, Sweden, 2008. Springer.

F. Castaño, A. Rossi, M. Sevaux, and N. Velasco. A column generation approach to extend lifetime in wireless sensor networks with coverage and connectivity constraints. *Computers & Operations Research*, 52:220–230, 2014.

F. Castaño, E. Bourreau, N. Velasco, A. Rossi, and M. Sevaux. Exact approaches for lifetime maximization in connectivity constrained wireless multi-role sensor networks. *European Journal of Operational Research*, 241(1):28–38, 2015.

F. Castao, A. Rossi, M. Sevaux, and N. Velasco. On the use of multiple sinks to extend the lifetime in connected wireless sensor networks. *Electronic Notes in Discrete Mathematics*, 41:77–84, 2013.

B. Cavdaroglu, E. Hammel, J. E. Mitchell, T. C. Sharkey, and W. A. Wallace. Integrating restoration and scheduling decisions for disrupted interdependent infrastructure systems. *Annals of Operations Research*, 203(1):279–294, 2013.

Y. Censor. Pareto optimality in multiobjective problems. *Applied Mathematics and Optimization*, 4 (1):41–59, 1977.

J. H. Chang and L. Tassiulas. Routing for maximum system lifetime in wireless ad hoc networks. In *Proceedings of the 37th Annual Allerton Conference on Communication, Control, and Computing*, volume 37, pages 1191–1200, Monticello, IL, 1999. University of Illinois at Urbana-Champaign.

J. H. Chang and L. Tassiulas. Maximum lifetime routing in wireless sensor networks. *IEEE/ACM Transactions on Networking*, 12(4):609–619, 2004.

B. Chen, R. Jiang, T. Kasetkasem, and P. K. Varshney. Channel aware decision fusion in wireless sensor networks. *IEEE Transactions on Signal Processing*, 52(12):3454–3458, 2004.

M. Chen, S. Gonzalez, A. Vasilakos, H. Cao, and V. C. M. Leung. Body area networks: A survey. *Mobile Networks and Applications*, 16(2):171–193, 2011.

W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. Maximum flow problems. *Combinatorial Optimization, First Edition*, pages 37–89, 1998.

R. M. Curry and J. C. Smith. A survey of optimization algorithms for wireless sensor network lifetime maximization. *Computers & Industrial Engineering*, 101:145–166, 2016.

J. C. Dagher, M. W. Marcellin, and M. A. Neifeld. A theory for maximizing the lifetime of sensor networks. *IEEE Transactions on Communications*, 55(2):323–332, 2007.

G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1): 101–111, 1960.

M. L. Das. Two-factor user authentication in wireless sensor networks. *IEEE Transactions on Wireless Communications*, 8(3):1086–1090, 2009.

I. R. de Farias, E. L. Johnson, and G. L. Nemhauser. Branch-and-cut for combinatorial optimization problems without auxiliary binary variables. *The Knowledge Engineering Review*, 16(1):25–39, 2001.

J. Desrosiers and M. E. Lübbecke. *A primer in column generation.* Springer US, 2005.

J. Desrosiers, Y. Dumas, M. M. Solomon, and F. Soumis. Time constrained routing and scheduling. In M. Ball, T. Magnanti, C. Monma, and G. Nemhauser, editors, *Handbooks in Operations Research and Management Science*, volume 8, pages 35–139. Elsevier, Amsterdam, 1995.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1: 269–271, 1959.

T. Ducrocq, N. Mitton, and M. Hauspie. Energy-based clustering for wireless sensor network lifetime optimization. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, pages 968–973, Shanghai, 2013. IEEE.

N. Dutta, A. Saxena, and S. Chellappan. Defending wireless sensor networks against adversarial localization. In Y. Lee and D. Lin, editors, *Proceedings of the 2010 Eleventh International Conference on Mobile Data Management*, Los Alamitos, CA, 2010. IEEE Computer Society.

J. Edmonds and R. M. Karp. Theoretical improvement in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

M. L. Fisher. An applications oriented guide to Lagrangian relaxation. *Interfaces*, 15(2):10–21, 1985.

L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

L. R. Ford and D. R. Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5(1):97–101, 1958.

G. A. Gabriele and K. M. Ragsdell. The generalized reduced gradient method: A reliable tool for optimal design. *Journal of Engineering for Industry*, 99(2):394–400, 1977.

S. R. Gandham, M. Dawande, R. Prakash, and S. Venkatesan. Energy efficient schemes for wireless sensor networks with multiple mobile base stations. In *Proceedings of the IEEE GLOBECOM '03*, pages 377–381, San Francisco, CA, 2003. IEEE.

M. R. Garey and D. S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.

N. Garg and J. Köneman. Faster and simpler algorithms for multi-commodity flow and other fractional packing problems. In *Proceedings of the 39th Annual Symposium of Foundations of Computer Science*, pages 300–309, Palo Alto, CA, November 1998.

M. Gatzianas and L. Georgiadis. A distributed algorithm for maximum lifetime routing in sensor networks with mobile sink. *IEEE Transactions on Wireless Communications*, 7(3):984–994, 2008.

M. Gentili and A. Raiconi. α-coverage to extend network lifetime on wireless sensor networks. *Optimization Letters*, 7(1):157–172, 2013.

G. Gerald, T. Fischer, T. Gally, A. M. Gleixner, G. Hendel, T. Koch, S. J. Maher, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, S. Schenker, R. Schwarz, F. Serrano, Y. Shinano, S. Vigerske, D. Weninger, M. Winkler, J. T. Witt, and J. Witzig. The SCIP optimization suite 3.2. Technical Report 15-60, ZIB, Takustr.7, 14195 Berlin, 2016.

J. M. Gilbert and F. Balouchi. Comparison of energy harvesting systems for wireless sensor networks. *International Journal of Automation and Computing*, 5(4):334–347, 2008.

B. Golden. A problem in network interdiction. *Naval Research Logistics Quarterly*, 25(4):711–713, 1978.

D. Goldfarb and W. Chen. On strongly polynomial dual simplex algorithms for the maximum flow problem. *Mathematical Programming*, 78(2):159–168, 1997.

D. Goldfarb and J. Hao. A primal simplex algorithm that solves the maximum flow problem in at most nm pivots and $o(n^2m)$ time. *Mathematical Programming*, 47(1–3):353–365, 1990.

J. S. He, S. Ji, Y. Pan, and Z. Cai. Approximation algorithms for load-balanced virtual backbone construction in wireless sensor networks. *Theoretical Computer Science*, 507:2–16, 2013.

W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1 (4):660–670, 2002.

W. R. Heinzelman, A. Chadrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, pages 1–10, Wailea Maui, HI, January 2000.

J. L. Higle and S. Sen. Stochastic decomposition: An algorithm for two-stage linear programs with recourse. *Mathematics of Operations Research*, 16(3):650–669, 1991.

D. S. Hochbaum. The pseudoflow algorithm: A new algorithm for the maximum-flow problem. *Operations Research*, 56(4):992–1009, 2008.

D. S. Hochbaum and A. Segev. Analysis of a flow problem with fixed charges. *Networks*, 19(3): 291–312, 1989.

Q. Huang, Y. Feng, X. Li, and B. Huang. A hexagonal grid based sink relocation method in wireless sensor networks. In *Proceedings of the Ninth International Conference on Frontier of Computer Science and Technology*, pages 76–80, Dalian, China, 2015. IEEE.

U. Janjarassuk and J. Linderoth. Reformulation and sampling to solve a stochastic network interdiction problem. *Networks*, 52(3):120–132, 2008.

N. Javaid, T. N. Qureshi, A. H. Khan, A. Iqbal, E. Akhtar, and M. Ishfaq. EDDEEC: Enhanced developed distributed energy-efficient clustering for heterogeneous wireless sensor networks. *Procedia Computer Science*, 19:914–919, 2013.

R. Kacimi, R. Dhaou, and A. L. Beylot. Load balancing techniques for lifetime maximizing in wireless sensor networks. *Ad Hoc Networks*, 11(8):2172–2186, 2013.

J. Kallrath. Mixed integer optimization in the chemical process industry: Experience, potential, and future perspectives. *Chemical Engineering Research and Design*, 78(6):809–822, 2000.

J. Kallrath. Combined strategic and operational planning–an MILP success story in the chemical industry. *OR Spectrum*, 24(3):315–341, 2002.

K. Kalpakis, K. Dasgupta, and P. Namjoshi. Efficient algorithms for maximum lifetime data gathering and aggregation in wireless sensor networks. *Computer Networks*, 42(6):697–716, 2003.

A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava. Power management in energy harvesting sensor networks. *ACM Transactions on Embedded Computer Systems*, 6(4), 2007.

J. Kennedy. Particle swarm optimization. In C. Sammut and G. I. Webb, editors, *Encyclopedia of Machine Learning*, pages 760–766. Springer, New York, 2011.

J. L. Kennington. Survey of linear cost multicommodity network flows. *Operations Research*, 26: 209–236, 1978.

M. E. Keskin, İ. K. Altınel, N. Aras, and C. Ersoy. Lifetime maximization in wireless sensor networks using a mobile sink with nonzero traveling time. *The Computer Journal*, 54(12):1987–1999, 2011.

A. W. Khan, A. H. Abdullah, M. H. Anisi, and J. I. Bangash. A comprehensive study of data collection schemes using mobile sinks in wireless sensor networks. *Sensors*, 14(2):2510–2548, 2014.

H. Kim, Y. Seok, N. Choi, Y. Choi, and T. Kwon. Optimal multi-sink positioning and energy-efficient routing in wireless sensor networks. In C. Kim, editor, *International Conference on Information Networking*, volume 3391, pages 264–274. Springer, 2005.

J. Ko, C. Lu, M. B. Srivastva, J. A. Stankovic, A. Terzis, and M. Welsh. Wireless sensor networks for healthcare. *Proceedings of the IEEE*, 98(11):1947–1960, 2010.

P. Kuila and P. K. Jana. Energy efficient clustering and routing algorithms for wireless sensor networks: Particle swarm optimization approach. *Engineering Applications of Artificial Intelligence*, 33: 127–140, 2014.

D. Kumar, T. C. Aseri, and R. B. Patel. EEHC: Energy Efficient Heterogeneous Clustered scheme for wireless sensor networks. *Computer Communications*, 32:662–667, 2009.

G. Laporte. Generalized subtour elimination constraints and connectivity constraints. *Journal of the Operational Research Society*, 37(5):509–514, 1986.

N. M. A. Latiff, C. C. Tsimenidis, and B. S. Sharif. Energy-aware clustering for wireless sensor networks using particle swarm optimization. In *Proceedings of the Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1–5, Athens, 2007. IEEE.

S. H. Lee, S. Lee, H. Song, and H. S. Lee. Wireless sensor network design for tactical military applications: Remote large-scale environments. In *Proceedings of 28th IEEE Conference on Military Communications*, pages 911–917, Boston, 2009. IEEE.

J. S. Leu, T. H. Chiang, M. C. Yu, and K. W. Su. Energy efficient clustering scheme for prolonging the lifetime of wireless sensor network with isolated nodes. *IEEE Communications Letters*, 19(2): 259–262, 2015.

J. Li and P. Mohapatra. An analytical model for the energy hole problem in many-to-one sensor networks. In *Proceedings of IEEE 62nd Semiannual Vehicular Technology Conference*, pages 2721–2725, Dallas, TX, 2005. IEEE.

J. Li and P. Mohapatra. Analytical modeling and mitigation techniques for the energy hole problem in sensor networks. *Pervasive and Mobile Computing*, 3(8):233–254, 2007.

K. Li, H. Luan, and C. C. Shen. Qi-ferry: Energy-constrained wireless charging in wireless sensor networks. In *Proceedings of the Wireless Communications and Networking Conference*, pages 2515–2520. IEEE, 2012.

M. Li, Z. Li, and A. V. Vasilakos. A survey on topology control in wireless sensor networks: Taxonomy, comparative study, and open issues. *Proceedings of the IEEE*, 101(12):2538–2557, 2013.

R. Li, X. Liu, W. Xie, and N. Huang. Deployment-based lifetime optimization model for homogeneous wireless sensor network under retransmission. *Sensors*, 14(12):23697–23723, 2014.

W. Liang and J. Luo. Network lifetime maximization in sensor networks with multiple mobile sinks. In *Proceedings of the IEEE 36th Conference on Local Computer Networks*, pages 350–357, Bonn, Germany, 2011. IEEE.

C. Liu and G. Cao. Distributed critical location coverage in wireless sensor networks with lifetime constraint. In *Proceedings of the 2012 IEEE INFOCOM*, pages 1314–1322, Orlando, FL, 2012. IEEE.

J. D. Lundquist, D. R. Cayan, and M. D. Dettinger. Meteorology and hydrology in Yosemite national park: A sensor network application. In *Proceedings of the 2nd International Symposium on Information Processing in Sensor Networks*, pages 518–528, Palo Alto, CA, 2003. Xerox PARC.

H. Luo, J. Luo, Y. Liu, and S. K. Das. Adaptive data fusion for energy efficient routing in wireless sensor networks. *IEEE Transactions on Computers*, 55(10):1286–1299, 2006.

R. Madan and S. Lall. Distributed algorithms for maximum lifetime routing in wireless sensor networks. In *Proceedings of the IEEE GLOBECOM '04*, pages 748–753, Dallas, TX, 2004. IEEE.

V. S. Mansouri, A. H. M. Rad, and V. W. S. Wong. Multicommodity lifetime routing for wireless sensor networks with multiple sinks. In *IEEE International Conference on Communications 2008*, Beijing, 2008. IEEE.

M. Marta and M. Cardei. Improved sensor network lifetime with multiple mobile sinks. *Journal of Pervasive and Mobile Computing*, 5(5):542–555, 2009.

G. P. McCormick. *Nonlinear Programming: Theory, Algorithms, and Applications.* John Wiley & Sons, Inc., New York, NY, USA, 1983.

C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the Association for Computing Machinery*, 7(4):326–329, 1960. ISSN 0004-5411.

J. Min, J. Kim, Y. Kwon, and Y. Lee. Multi-channel MAC protocol for real-time monitoring of weapon flight test in wireless sensor network. In S. Yurish, editor, *Proceedings of the Sixth International Conference on Sensor Technologies and Applications*, pages 83–88, Rome, 2012. IARIA.

N. Mladenović, J. Brimberg, P. Hansen, and J. A. Moreno-Pérez. The p-median problem: A survey of metaheuristic approaches. *European Journal of Operational Research*, 179(3):927–939, 2007.

A. B. Mohanoor, S. Radhakrishnan, and V. Sarangan. Online energy aware routing in wireless networks. *Ad Hoc Networks*, 7(5):918–931, 2009.

S. Monsell. Task switching. *Trends in Cognitive Sciences*, 7(3):134–140, 2003.

S. Movassaghi, M. Abolhasan, J. Lipman, D. Smith, and A. Jamalipour. Wireless body area networks: A survey. *IEEE Communications Surveys and Tutorials*, 16(3):1658–1686, 2014.

J. M. Mulvey and A. Ruszczyński. A new scenario decomposition method for large-scale stochastic optimization. *Operations Research*, 43(3):477–490, 1995.

J. M. Mulvey, R. J. Vanderbei, and S. A. Zenios. Robust optimization of large-scale systems. *Operations Research*, 43(2):264–281, 1995.

S. D. Muruganathan, D. C. F. Ma, R. I. Bhasin, and A. O. Fapojuwo. A centralized energy-efficient routing protocol for wireless sensor networks. *IEEE Radio Communications*, 43(3):S8–13, 2005.

M. J. Neely. Stochastic network optimization with application to communication and queueing systems. *Synthesis Lectures on Communication Networks*, 3(1):1–211, 2010.

S. A. Nikolidakis, D. Kandris, D. D. Vergados, and C. Douligeris. Energy efficient routing in wireless sensor networks through balanced clustering. *Algorithms*, 6(1):29–42, 2013.

F. Ordóñez and B. Krishnamachari. Optimal information extraction in energy-limited wireless sensor networks. *IEEE Journal on Selected Areas in Communications*, 22(6):1121–1129, 2004.

S. Özdemir, B. A. Attea, and O. A. Khalil. Multi-objective evolutionary algorithm based on decomposition for energy efficient coverage in wireless sensor networks. *Wireless Personal Communications*, 71:195–215, 2013.

N. A. Pantazis, S. A. Nikolidakis, and D. D. Vergados. Energy-efficient routing protocols in wireless sensor networks: A survey. *IEEE Communications Surveys & Tutorials*, 15(2):551–591, 2013.

I. Papadimitriou and L. Georgiadis. Energy-aware routing to maximize lifetime in wireless sensor networks with mobile sink. *Journal of Communications Software and Systems*, 2(2):141–151, 2006.

J. Park and S. Sahni. An online heuristic for maximum lifetime routing in wireless sensor networks. *IEEE Transactions on Computers*, 55(8):1048–1056, 2006.

T. Peltola, J. Matakselkä, E. Harju, H. Salovuori, J. Kenskinen, K. Mäkinen, and O. Roikonen. Method for congestion management in a frame relay network and a node in a frame relay network, June 10 1997. U.S. Patent 5,638,359.

A. F. Perold. Large-scale portfolio optimization. *Management Science*, 30:1143–1160, 1984.

A. Perrig, J. Stankovic, and D. Wagner. Security in wireless sensor networks. *Communications of the ACM*, 47(6):53–57, 2004.

G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

P. M. Pradhan and G. Panda. Connectivity constrained wireless sensor deployment using multi-objective evolutionary algorithms and fuzzy decision making. *Ad Hoc Networks*, 10(6):1134–1145, 2012.

A. P. Punnen. A linear time algorithm for the maximum capacity path problem. *European Journal of Operational Research*, 53(3):402–404, 1991.

A. Quelhas, E. Gil, J. D. McCalley, and S. M. Ryan. A multiperiod generalized network flow model of the U.S. integrated energy systems: Part I–Model description. *IEEE Transactions on Power Systems*, 22(2):829–836, 2007.

C. S. Raghavendra, K. M. Sivalingam, and T. F. Znati. *Wireless Sensor Networks*. Springer, New York City, 2004.

T. Rault, A. Bouabdallah, and Y. Challal. Energy efficiency in wireless sensor networks: A top-down survey. *Computer Networks*, 67:104–122, 2014.

P. Rawat, K. D. Singh, H. Chaouchi, and J. M. Bonnin. Wireless sensor networks: A survey on recent developments and potential synergies. *The Journal of Supercomputing*, 68(1):1–48, 2014.

S. Rizvi, H. K. Qureshi, S. A. Khayam, V. Rakocevic, and M. Rajarajan. A1: An energy efficient topology control algorithm for connected area coverage in wireless sensor networks. *Journal of Network and Computer Applications*, 35(2):597–605, 2012.

A. Romich, G. Lan, and J. C. Smith. A robust sensor covering and communication problem. *Naval Research Logistics*, 62(7):582–594, 2015.

A. Rossi, A. Singh, and M. Sevaux. An exact approach for maximizing the lifetime of sensor networks with adjustable sensing ranges. *Computers & Operations Research*, 39(12):3166–3176, 2012a.

A. Rossi, A. Singh, and M. Sevaux. Column generation algorithm for sensor coverage scheduling under bandwidth constraints. *Networks*, 60(3):141–154, 2012b.

L. B. Saad and B. Tourancheau. Multiple mobile sinks positioning in wireless sensor networks for buildings. In *Proceedings of the 2009 Third International Conference on Sensor Technologies and Applications*, pages 264–270, Athens, 2009. IEEE.

G. S. Sara and D. Sridharan. Routing in mobile wireless sensor network: A survey. *Telecommunication Systems*, 57(1):51–79, 2014.

P. Schaffer, K. Farkas, Á. Horváth, T. Holczer, and L. Buttyán. Secure and reliable clustering in wireless sensor networks: A critical survey. *Computer Networks*, 56(11):2726–2741, 2012.

A. Schrijver. On the history of the transportation and maximum flow problems. *Mathematical Programming*, 91(3):437–445, 2002.

S. Sengupta, S. Das, M. D. Nasir, A. V. Vasilakos, and W. Pedrycz. An evolutionary multi-objective sleep-scheduling scheme for differentiated coverage in wireless sensor networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 42(6):1093–1102, 2012.

S. Sengupta, S. Das, M. D. Kasir, and B. K. Panigrahi. Multi-objective node deployment in WSNs: In search of an optimal trade-off among coverage, lifetime, energy consumption, and connectivity. *Engineering Applications of Artificial Intelligience*, 26:405–416, 2013.

V. Shah-Mansouri and V. W. S. Wong. Distributed maximum lifetime routing in wireless sensor networks based on regularization. In *Proceedings of the Global Telecommunications Conference*, pages 598–603, Washington, D.C., 2007. IEEE.

F. Shan, W. Liang, J. Luo, and X. Shen. Network lifetime maximization for time-sensitive data gathering in wireless sensor networks with a mobile sink. *Computer Networks*, 57(7):1063–1077, 2013.

H. D. Sherali and J. C. Smith. Improving discrete model representations via symmetry considerations. *Management Science*, 47(10):1396–1407, 2001.

E. Shi and A. Perrig. Designing secure sensor networks. *IEEE Wireless Communications*, 11(6): 38–43, 2004.

B. Singh and D. K. Lobiyal. A novel energy-aware cluster head selection based on particle swarm optimization for wireless sensor networks. *Human-centric Computing and Information Sciences*, 2 (13):1–8, 2012.

J. C. Smith, M. Prince, and J. Geunes. Modern network interdiction problems and algorithms. In P. M. Pardalos, D. Du, and R. Graham, editors, *Handbook of Combinatorial Optimization*, pages 1949–1987. Springer, 2013.

S. Tao, K. Xu, Y. Xu, T. Fei, L. Gao, R. Guerin, J. Kurose, D. Towsley, and Z. Zhang. Exploring the performance benefits of end-to-end path switching. In *Proceedings of the 12th IEEE International Conference on Network Protocols*, pages 304–315, Berlin, 2004. IEEE.

C. H. Timpe and J. Kallrath. Optimal planning in large multi-site production networks. *European Journal of Operational Research*, 126(2):422–435, 2000.

C. Toh, H. Cobb, and D. Scott. Performance evaluation of battery-life-aware routing optimization schemes for wireless ad hoc networks. In *Proceedings of the IEEE International Computing Conference*, volume 9, pages 2824–2829. IEEE, 2011.

C. Tunca, S. Isik, M. Y. Donmez, and C. Ersoy. Distributed mobile sink routing for wireless sensor networks: a survey. *IEEE Communications Surveys & Tutorials*, 16(2):877–897, 2014.

Y. B. Türkoğullari, N. Aras, I. K. Altinel, and C. Ersoy. A column generation based heuristic for sensor placement, activity scheduling, and data routing in wireless sensor networks. *European Journal of Operational Research*, 207(2):1014–1026, 2010.

H. J. Visser and R. J. M. Vullers. RF energy harvesting and transport for wireless sensor network applications: Principles and requirements. *Proceedings of the IEEE*, 101(6):1410–1423, June 2013.

R. Vullers, R. Schaijk, H. Visser, J. Penders, and C. Hoof. Energy harvesting for autonomous wireless sensor networks. *IEEE Solid-State Circuits Magazine*, 2(2):29–38, 2010.

A. Wahid, S. Lee, and D. Kim. A reliable and energy-efficient routing protocol for underwater wireless sensor networks. *International Journal of Communication Systems*, 27:2048–2062, 2014.

L. Wang and X. Wu. Distributed prevention mechanism for network partitioning in wireless sensor networks. *Journal of Communications and Networks*, 16(6):667–676, 2014.

Z. M. Wang, S. Basagni, E. Melachrinoudis, and C. Petrioli. Exploiting sink mobility for maximizing sensor network lifetime. In *Proceedings of the 38th Annual Hawaii International Conference on System Science*, volume 9, pages 287a–287a, Waikoloa, HI, 2005. IEEE.

G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Walsh. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing, Special Issue on Data-Driven Applications in Sensor Networks*, 10:18–25, 2006.

R. D. Wollmer. Maximizing flow through a network with node and arc capacities. *Transportation Science*, 2(3):213–232, 1968.

G. Xu, W. Shen, and X. Wang. Applications of wireless sensor networks in marine environment monitoring: A survey. *Sensors*, 14(9):16932–16954, 2014.

Z. Xu, W. Lian, and Y. Xu. Network lifetime maximization in delay-tolerant sensor networks with a mobile sink. In *Proceedings of the 8th IEEE International Conference on Distributed Computing in Sensor Systems*, pages 9–16, Hangzhou, China, 2012. IEEE Computer Society.

Y. Xue, Y. Cui, and K. Nahrstedt. Maximizing lifetime for data aggregation in wireless sensor networks. *Mobile Networks and Applications*, 10(6):853–864, 2005.

J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12): 2292–2330, 2008.

M. Younis, I. F. Senturk, K. Akkaya, S. Lee, and F. Senel. Topology management techniques for tolerating node failures in wireless sensor networks: A survey. *Computer Networks*, 58:254–283, 2014.

S. Yu, B. Zhang, C. Li, and H. T. Mouftah. Routing protocols for wireless sensor networks with mobile sinks: A survey. *IEEE Communications Magazine*, 52(7):150–157, 2014.

Y. Yun and Y. Xia. Maximizing the lifetime of wireless sensor networks with mobile sink in delay-tolerant applications. *IEEE Transactions on Mobile Computing*, 9(9):1308–1318, 2010.

Y. Yun, Y. Xia, B. Behdani, and J. C. Smith. Distributed algorithm for lifetime maximization in delay-tolerant wireless sensor network with mobile sink. In *Proceedings of the 49th IEEE Conference on Decision and Control (CDC)*, pages 370–375, Atlanta, GA, 2010a. IEEE.

Z. Yun, X. Bai, D. Xuan, T. H. Lai, and W. Jia. Optimal deployment patterns for full coverage and k-Connectivity ($k \leq 6$) wireless sensor networks. *IEEE/ACM Transactions on Networking*, 18(3): 934–937, 2010b.

B. Zhang, Z. Jiao, C. Li, Z. Yao, and A. V. Vasilakos. Efficient location-based topology control algorithms for wireless ad hoc and sensor networks. *Wireless Communications and Mobile Computing*, 2016.

Y. Zhao, J. Wu, F. Li, and S. Lu. On maximizing the lifetime of wireless sensor networks using virtual backbone scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 23(8):1528–1535, 2011.