ALGORITHMS FOR SOLVING MULTI-LEVEL OPTIMIZATION PROBLEMS WITH DISCRETE VARIABLES AT MULTIPLE LEVELS

By

Z. CANER TAŞKIN

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2009

To my wife, my parents and my brother; I owe them everything I have

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to Cole Smith for his wise, enlightening ideas, endless motivation, and patient counseling during the writing of this dissertation. He has been a great teacher, a mentor and a friend to me in the last four years, and working with him has been a privilege.

I would like to thank Edwin Romeijn for introducing me to the exciting field of optimization in health care, and his invaluable contributions to this study. His guidance and support has been very helpful throughout my graduate education. I would also like to acknowledge Panos Pardalos and Douglas Dankel for taking part in my dissertation committee, and their valuable suggestions.

My sincere thanks are due to my friends in graduate school. In particular, Chase has not only been a great colleague, but also a close friend for these four years. I cannot begin to count the things that I learned from him about the culture, the lifestyle and the language. My experience in America would not have been nearly as enjoyable without him and his wife Candace.

I am indebted to my parents and my brother for guiding and supporting me in all life choices I have made. Finally, I am deeply grateful to my lovely wife, Semra, for her constant encouragement, support, and understanding. She is the source of my happiness, and the secret of my success. This dissertation represents the end of my life as a graduate student. It also represents the beginning of a new stage in my life, every moment of which I am looking forward to sharing with her.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

ALGORITHMS FOR SOLVING MULTI-LEVEL OPTIMIZATION PROBLEMS WITH DISCRETE VARIABLES AT MULTIPLE LEVELS

By

Z. Caner Taşkın

August 2009

Chair: J. Cole Smith
Major: Industrial and Systems Engineering

In this dissertation, we investigate a class of multi-level optimization problems, in which discrete variables are present at multiple stages. Such problems arise in many practical settings, and they are notoriously difficult to optimize. Benders decomposition, which is a well-known decomposition method for solving large-scale mixed-integer programming problems, cannot be utilized for the class of problems that we consider due to the existence of discrete variables at lower levels. Cutting plane algorithms such as those proposed by Laporte and Louveaux have been designed for use in bi-level integer programming problems with binary variables at both levels. However, these are based on generic cuts, which do not utilize any problem specific structures, and hence often result in weak convergence. Our goal in this dissertation is to propose novel formulation and solution strategies for several multi-level optimization problems to solve these problems to optimality within practical computational limits.

We first consider an edge-partition problem. The motivation for this study is provided by a Synchronous Optical Network (SONET) design application. In the SONET context, each edge represents a demand pair between two client nodes, and the weight of each edge represents the number of communication channels needed between the client nodes. We consider a stochastic version of the problem, in which the edge weights are not deterministic, but their underlying probability distribution is known. The problem is to design a set of SONET "rings" at minimum cost, while ensuring that

the resulting network can handle the random demand with a prespecified level of service. We first model the problem as a large-scale integer program, and attempt to solve it via a bi-level decomposition approach, in which both levels contain binary variables. We then propose a three-level solution approach for the problem, which is based on a hybrid integer programming/constraint programming decomposition algorithm. We show computationally that the hybrid algorithm significantly outperforms the other approaches.

Next, we focus on a matrix decomposition problem, which arises in Intensity Modulated Radiation Therapy (IMRT) treatment planning. The problem input is a matrix of intensity values that needs to be delivered to a patient, which must be decomposed into a collection of apertures and corresponding intensities. In a feasible decomposition the sum of binary shape matrices multiplied by corresponding intensity values is equal to the original intensity matrix. We consider two variants of the problem: (i) a variant in which the shape matrices used in the decomposition have to satisfy the "consecutive-ones" property, and (ii) a variant in which the shape matrices have to be rectangular. For the first variant, we start by investigating an integer programming model proposed in the literature, and show how the formulation can be strengthened. We then formulate the problem as a bi-level optimization problem that has discrete variables at both stages, and suggest a hybrid integer programming/constraint programming decomposition algorithm similar to our algorithm for the edge-partition problem. Our tests on data obtained from patients show that our algorithm is capable of solving problem instances of clinically relevant dimensions within practical computational limits. We then turn our attention to the second variant of the matrix decomposition problem. We formulate the problem as a mixed-integer program, and investigate a decomposition method for solving it. Unlike the first variant, the second-level problem turns out to be a linear programming problem, and therefore we are able to derive a Benders decomposition algorithm for solving this variant.

Finally, we investigate a class of graph search problems. In this class of problems, an intruder is located at an unknown node on the input graph, and a group of searchers needs

to be coordinated to detect the intruder within a limited amount of time. This problem arises in settings such as search-and-rescue and search-and-capture operations, and patrol route design. We investigate three variants of the problem: (i) a hide-and-seek version, in which a stationary intruder is hiding at an unknown node, (ii) a pursuit evasion version, in which the intruder can move across the edges of the graph to avoid being detected, and (iii) a patrol problem, in which the searchers are assigned to recurring patrol routes to protect the graph from intrusion. We model each problem as a large-scale integer program, and propose a branch-cut-price algorithm to find the minimum number of searchers needed, and a route for each searcher. In our formulation, both the master problem and the subproblems corresponding to the searchers and the intruder contain binary variables. We model the master problem as a set covering problem and propose a solution approach that is based on dynamic column and row generation.

CHAPTER 1
INTRODUCTION

In most complex decision-making environments, there exist several types of interdependent decisions that need to be made to optimize some cost or benefit function. As a simple example, consider a production planning problem. The goal is to determine, at the very least, the types of products that are to be produced within a time period, along with the associated production quantities. There might exist individual restrictions on each type of decision, such as "the total amount of production of products A and B cannot exceed α," or "if product A is produced, so must product B." There might also exist restrictions that relate the two types of decisions, such as "if product A is produced, then the batch size must be at least β." Modeling an optimization problem involves (i) defining a decision variable for each individual decision, (ii) formulating the restrictions on the decisions as constraints, and (iii) defining an objective function to be optimized. The field of mathematical programming seeks to optimize such models and prove the optimality of the generated solution, or prove that no feasible solution exists.

Optimization problems in which some variables are restricted to take values from a discrete set are classified as discrete optimization problems. An important concern regarding building and solving discrete optimization problems is that the amount of memory and the computational effort needed to solve such problems grow exponentially with the number of discrete variables. The traditional approach, which involves making all decisions simultaneously by solving a monolithic optimization problem, quickly becomes intractable as the number of discrete variables increases. Multi-level optimization algorithms, such as Benders decomposition (Benders, 1962), have been developed as an alternative solution methodology to alleviate this difficulty. Unlike the traditional approach, these algorithms divide the decision-making process into several stages. For instance, in Benders decomposition a first-stage master problem is solved for a subset of variables, and the values of the remaining variables are determined by a second-stage

subproblem *given* the values of the first-stage variables. In our simplistic production planning example, the master problem selects a subset of the products to be produced, by considering only the restrictions such as "if product A is produced, so must product B." Then, given the set of selected products, a subproblem seeks the production quantities considering more detailed production restrictions such as "the total amount of production of products A and B cannot exceed α," and "if a product is produced, then the batch size must be at least β." If the subproblem is able to find a solution for the second-stage variables, then a solution for the overall problem can be obtained by combining the first- and second-stage decisions. However, the subproblem can also determine that the current selection of products by the master problem does not yield a feasible solution when the detailed production constraints are considered. In this case, a constraint that eliminates the current selection of products from further consideration is added to the master problem, which is then re-solved. In this manner, the optimization problem is solved via a series of "solution proposals" made by the master problem, and "reasons for rejection" by the subproblem. This iterative algorithm eventually converges to an optimal solution for the overall problem.

In essence, multi-level optimization algorithms solve a series of small problems instead of a single large problem. Performing multiple iterations is usually justified due to the exponentially larger computational resource requirements associated with solving a larger problem. Furthermore, it is often the case that decisions for several groups of second-stage variables can be made independently given the first-stage decisions. In such cases, multi-level optimization algorithms are amenable to parallel implementations. The advent of efficient parallel computing grids has allowed modern bi-level techniques to solve problems that were regarded as intractable before (Ntaimo and Sen, 2005). In some applications, solving problems in multiple stages allows effort to be conserved by avoiding the explicit solution of problems by mathematical programming, such as the evacuation network design algorithm of Andreas and Smith (2009). Multi-level optimization has

recently received much attention due to the emerging importance of research in fields like stochastic programming and network interdiction. Many problems can be formulated naturally as multi-level optimization problems (Migdalas and Pardalos, 1996; Migdalas et al., 1997), and a wide class of optimization problems can be reformulated as multi-level optimization problems (Huang and Pardalos, 2002).

Benders decomposition has been particularly successful in solving mixed-integer linear programming problems arising in a wide variety of applications. In Benders decomposition, discrete variables of the problem are kept in the master problem, and continuous variables are moved to the subproblem. In each iteration, given the values of the discrete variables by the master problem, the subproblem is solved as a linear program, and a cutting plane to be passed back to the master problem is generated using linear programming duality. However, this approach cannot be applied directly when discrete variables also appear in the second stage. The reason is that no dual information can be extracted from the subproblem if the second-stage problem contains integer variables. In this case, one can employ cutting planes such as the general-purpose cuts of Laporte and Louveaux (1993) and combinatorial Benders inequalities (Codato and Fischetti, 2006). However, these inequalities are often very weak, and result in slow algorithmic convergence.

Our main line of research is about designing efficient multi-level optimization algorithms for problems that have discrete variables at multiple stages. We first present our results on an edge-partition problem arising in a telecommunications network design context regarding Synchronous Optical Networks (SONET). The edge-partition problem considers an undirected graph with weighted edges, and simultaneously assigns nodes and edges to subgraphs such that each edge appears in exactly one subgraph, and such that no edge is assigned to a subgraph unless both of its incident nodes are also assigned to that subgraph. Additionally, there are limitations on the number of nodes and on the sum of edge weights that can be assigned to each subgraph (Goldschmidt et al., 2003).

We consider a stochastic version of the edge-partition problem in Chapter 2, in which we assign nodes to subgraphs in a first stage, realize a set of edge weights from among a finite set of alternatives, and then assign edges to subgraphs. We first formulate the problem as a monolithic integer programming problem, and show that this approach is not tractable due to the rapidly increasing computational requirements. We then prescribe a bi-level cutting plane approach having integer variables in both stages and examine computational difficulties associated both with the generic inequalities by Laporte and Louveaux (1993) and with our proposed cutting planes. We also prescribe a three-level hybrid integer programming/constraint programming algorithm having discrete variables at all levels, and discuss how this hybrid approach resolves some of the difficulties associated with the bi-level cutting plane approach.

Chapters 3 and 4 consider a problem dealing with the efficient delivery of Intensity Modulated Radiation Therapy (IMRT) to individual patients. In particular, we consider a matrix decomposition problem that arises at the leaf sequencing stage in IMRT treatment planning. The problem input is an integer matrix of intensity values that are to be delivered to a patient from a given beam angle. To deliver this intensity profile to the patient, we must decompose the input matrix into a collection of apertures and corresponding intensities. A feasible decomposition is one in which the original desired intensity profile matrix is equal to the sum of a number of feasible binary matrices multiplied by corresponding intensity values. To most efficiently treat a patient, we wish to minimize a measure of total treatment time, which is given as a weighted sum of the number of apertures and the sum of the aperture intensities used in the decomposition. In Chapter 3, we describe a version of the problem in which each aperture matrix needs to satisfy the "consecutive-ones" property, which means that all matrix entries with value 1 in each row of an aperture matrix must be consecutive. Similar to Chapter 2, we prescribe a bi-level hybrid optimization algorithm in which the master problem is an integer programming problem, and we solve a subproblem for each row of the matrix by

a constraint programming-based backtracking algorithm. Chapter 4 deals with another variant of the matrix decomposition problem, in which only rectangular apertures can be used in the decomposition. We develop a Benders decomposition algorithm for solving this variant. We also propose a scheme for partitioning the problem to obtain simultaneous upper and lower bounds.

In Chapter 5, we study a class of graph search problems, where a group of searchers seek an intruder on a graph. Both the intruder and the searchers are located at some nodes of the graph, and the searcher can only "see" a subset of the nodes from each node. At each time period, both the intruder and the searchers can move along an edge to an adjacent node, or stay at the same node. Our goal is to find the minimum number of searchers needed to locate the intruder within a given time limit. We investigate three variants of the graph search problem: (i) a hide-and-seek problem, in which a stationary intruder "hides" at an unknown node, (ii) a pursuit evasion problem, in which the intruder moves among the nodes to avoid being detected, and (iii) a patrol problem, in which no intruder is initially in the graph and each searcher patrols the graph to protect it from potential intrusion. We formulate these problems as a set covering problem with an exponential number of variables and constraints, and propose a branch-cut-price algorithm for solving it. Both our master problem and the subproblems, which correspond to the intruder and the searchers, have discrete variables. We formulate the intruder's subproblem as a longest path problem on an auxiliary graph, and the searcher's subproblems as mixed-integer programming problems.

We conclude our dissertation in Chapter 6, which explores future research directions regarding multi-level optimization algorithms. We first evaluate the approaches taken in the edge-partition, matrix decomposition, and graph search problems described in Chapters 2–5, and discuss future research topics regarding each application. We then describe our preliminary research on a master problem reformulation technique, which can be used in a variety of bi-level optimization algorithms. This reformulation technique

can eliminate an exponential number of iterations by adding a quadratic number of variables to the master problem. Therefore, it has the potential of leading to significant improvements in solvability of a class of bi-level optimization problems.

CHAPTER 2
STOCHASTIC EDGE-PARTITION PROBLEM

2.1 Introduction and Literature Survey

We begin by describing the *edge-partition problem* of Goldschmidt et al. (2003), which is defined on an undirected graph $G(N, E)$. In the deterministic edge-partition problem, we create a set K of (possibly empty) subgraphs of G such that each edge is contained in exactly one subgraph, subject to certain restrictions on the composition of the subgraphs. These restrictions include the stipulations that an edge cannot be assigned to a subgraph unless both of its incident nodes belong to the subgraph, and that no more than r nodes can be assigned to any subgraph, for some $r \in \mathbb{Z}^+$. Additionally, each edge $(i, j) \in E$ has a positive weight of w_{ij}, and the sum of edge weights assigned to each subgraph cannot exceed some given positive number b. The objective of the problem is to minimize the sum of the number of nodes assigned to each subgraph.

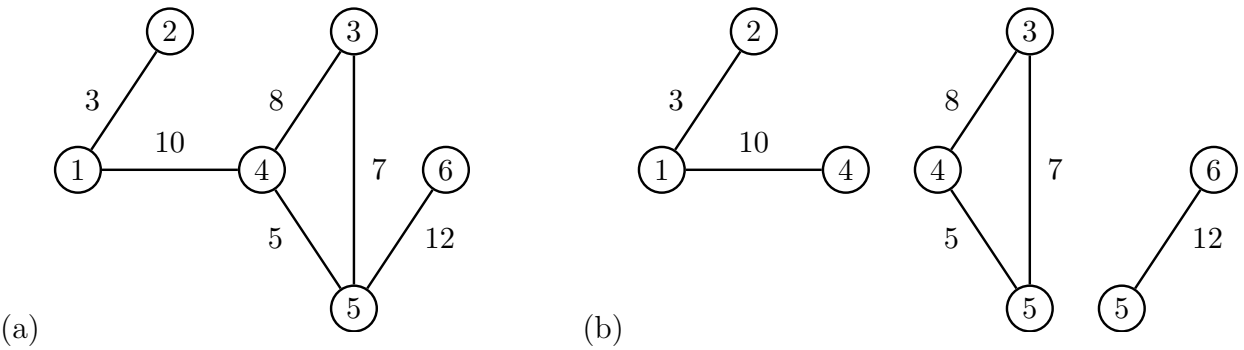


Figure 2-1. (a) An instance of the deterministic edge-partition problem (b) A solution with $|K| = 3,\ r = 3,\ b = 20$

Figure 2-1 illustrates the deterministic edge-partition problem. The graph G and the corresponding edge weights are shown in Figure 2-1a. Figure 2-1b shows a feasible solution that partitions G into $|K| = 3$ subgraphs, where the number of nodes in each subgraph is limited by $r = 3$, and the total weight assigned to each subgraph is limited by $b = 20$. Note that the degree of node 4 is three, which implies that it must be assigned to at least two subgraphs, or else there would be at least $4 > r$ nodes in a subgraph. Similarly, node 5 must be assigned to at least two subgraphs. Since nodes 4 and 5 are assigned to two

subgraphs, and every other node is assigned to a single subgraph, the solution represented by Figure 2-1b is optimal.

Goldschmidt et al. (2003) discuss the edge-partition problem (with deterministic weights) in the context of designing Synchronous Optical Network (SONET) rings. In the SONET context, each edge $(i, j) \in E$ represents a demand pair between two client nodes, and the weight w_{ij} represents the number of communication channels requested between nodes i and j. All telecommunication traffic is carried over a set of SONET rings, which are represented by subgraphs in the edge-partition problem. Since each demand must be carried by exactly one ring, edges must be partitioned among the rings. (Note that the term "ring" describes only the physical SONET routing structure, and does not place any restrictions on topological properties of demand edges included on a ring. See, e.g., Goldschmidt et al. (2003) for more details.) SONET rings are permitted to carry communication between nodes only if those nodes have been connected to the ring by equipment called Add-Drop Multiplexers (ADMs). There are technical limits on the number of ADMs that can be assigned to each ring (e.g., r), and on the total amount of channels (e.g., b) that can be assigned to a ring. Since ADMs are quite expensive, ring networks are preferred that employ as few ADMs as possible, which echoes the edge-partition problem's objective of minimizing the sum of nodes assigned to each subgraph.

The primary contribution by Goldschmidt et al. (2003) is an approximation algorithm for a specific case of the edge-partition problem in which all w_{ij} are equal to one. Sutter et al. (1998) propose a column-generation algorithm for this problem, and Lee et al. (2000b) employ a branch-and-cut algorithm on a formulation that we adapt. For the case in which the weights on the edges can be split among multiple rings, Sherali et al. (2000) prescribe a mixed-integer programming approach augmented by the use of valid inequalities, anti-symmetry constraints, and variable branching rules. Smith (2005) formulates the deterministic version of the edge-partition problem as a constraint program

and examines several issues regarding symmetry and search algorithm design. Specifically, she shows how adding aggregate variables that represent the number of node copies (similar to our approach in Section 2.3) can improve performance.

We consider a version of the edge-partition problem where the edge weights are uncertain, and are only realized after the node-to-subgraph assignments have been made. As we show in Section 2.2, this framework allows us to design more robust solutions than those in the literature, which are virtually all applied to deterministic data. We seek a minimum-cardinality set of node-to-subgraph assignments, such that there exists an assignment of edges to subgraphs satisfying the aforementioned subgraph restrictions with a pre-specified high probability. Such a probabilistic constraint is extremely hard to deal with in an optimization framework. One approach, known as *scenario approximation* (cf. Calafiore and Campi (2005); Luedtke and Ahmed (2008); Nemirovski and Shapiro (2005)) is to draw independent identically distributed (i.i.d.) realizations of the edge weights (called scenarios) and require the node-to-subgraph assignments to admit a feasible edge-to-subgraph assignment in each scenario. It can be shown that, with a sufficiently large scenario set, a solution to this scenario approximation solution is feasible to the true probabilistically constrained problem with high confidence. In this study we develop algorithmic approaches for solving the scenario approximation corresponding to the discussed probabilistically constrained edge-partition problem. We refer to this scenario approximation as the *stochastic edge-partition problem*.

Relatively little work has been done in SONET network design when the edge weights are uncertain. Smith et al. (2004) consider the SONET ring design problem in which edge demands can be split among multiple rings and propose a two-stage integer programming algorithm. The demand splitting relaxation allows the second-stage problems to be solved as linear programs, and thus standard Benders cuts can easily be derived from the second-stage recourse problems. However, we have second-stage integer programs from which dual information cannot be readily obtained.

The edge-partition problem can also be approached using stochastic integer programming theory. For problems having binary first-stage variables and mixed-integer second-stage variables, such as our problem, a well-known decomposition approach is the integer L-shaped method (Laporte and Louveaux, 1993). This method approximates the second-stage value function by linear "cuts" that are exact at the binary solution where the cut is generated, and are under-estimates at other binary solutions. Typical integer programming algorithms progress by solving a sequence of intermediate linear programming (LP) problems. Using disjunctive programming techniques, it is possible to derive cuts from the solutions to these intermediate LPs that are valid under-estimators of the second-stage value function at all binary first-stage solutions (Sen and Higle, 2005; Sherali and Fraticelli, 2002). This avoids solving difficult integer second-stage problems to optimality in all iterations of the algorithm, providing significant computational advantage. Scenario-wise decomposition methods have also been proposed (Carøe and Schultz, 1999) as an alternative to the above stage-wise decomposition approaches. Here copies of the first-stage variables are made corresponding to each scenario and are linked together via non-anticipativity constraints.

Our proposed methodology draws on constraint programming and stochastic integer programming theory. Hybrid algorithms of this nature have recently been successfully employed to solve notoriously difficult problems. Jain and Grossmann (2001) and Bockmayr and Pisaruk (2006) devise hybrid integer programming/constraint programming algorithms for solving machine scheduling problems. Thorsteinsson (2001) proposes a framework for integrating integer programming and constraint programming approaches. Hooker and Ottosson (2003) extend the Benders decomposition framework so that constraint logic programs can be used as subproblems to generate cuts that are added to a mixed-integer linear master problem. A recent work by Hooker (2007) uses logic-based Benders decomposition to solve several planning and scheduling problems.

The remainder of this chapter is organized as follows. In Section 2.2, we develop a mixed-integer programming formulation for the stochastic edge-partition problem, and provide cutting planes that can be used within a two-stage decomposition algorithm. In Section 2.3, we prescribe an alternative three-stage algorithm to overcome the computational difficulties associated with the weakness of the proposed cutting planes. Finally, we compare the efficacy of these algorithms in Section 2.4 on a set of randomly generated test instances.

2.2 Formulation and Cutting Plane Approach

Let us introduce binary decision variables $x_{ik} = 1$ if node i is assigned to subgraph k and 0 otherwise, $\forall i \in N,\ k \in K$. For this formulation, we specify a value of $|K|$ that is sufficiently large to ensure that a feasible solution exists to the problem (as discussed in Section 2.4). We denote the vector of node-to-subgraph assignments by $\mathbf{x}$. Let $\tilde{\mathbf{w}}$ denote the random vector of edge weights with known distribution, and $\mathbf{w}$ denote a realization with components w_{ij}. We define binary decision variables $y_{ijk} = 1$ if edge (i, j) is assigned to subgraph k. Given an allowed violation probability $\epsilon \in (0, 1)$ the probabilistic edge-partition problem can be formulated as follows:

$$\text{Minimize} \sum_{i \in N} \sum_{k \in K} x_{ik} \tag{2–1}$$

$$\text{subject to} \sum_{i \in N} x_{ik} \leq r \quad \forall k \in K \tag{2–2}$$

$$x_{ik} \in \{0, 1\} \quad \forall i \in N,\ k \in K \tag{2–3}$$

$$\Pr\{G(\mathbf{x}, \tilde{\mathbf{w}}) \leq b\} \geq 1 - \epsilon, \tag{2–4}$$

where

$$G(\mathbf{x}, \mathbf{w}) = \text{Minimize}\ \ z \tag{2–5}$$

$$\text{subject to} \sum_{k \in K} y_{ijk} = 1 \quad \forall (i, j) \in E, \tag{2–6}$$

$$\sum_{(i,j)\in E} w_{ij} y_{ijk} \leq z \quad \forall k \in K \tag{2–7}$$

$$y_{ijk} \leq x_{ik},\ y_{ijk} \leq x_{jk} \quad \forall (i,j) \in E,\ k \in K \tag{2–8}$$

$$y_{ijk} \in \{0,1\} \quad \forall (i,j) \in E,\ k \in K. \tag{2–9}$$

The objective (2–1) minimizes the total number of nodes assigned to subgraphs. Constraints (2–2) limit the number of nodes assigned to each subgraph. Constraints (2–6) require that the edges be partitioned among the subgraphs. Constraints (2–7) compute the maximum assigned weight over all subgraphs. Constraints (2–8) require that no edge can be assigned to a subgraph unless both of its incident nodes are assigned to that subgraph, and (2–3) and (2–9) state logical restrictions on the variables. By convention, the optimal value $G(\mathbf{x}, \mathbf{w})$ of the integer program (2–5)–(2–9) is $+\infty$ if the problem is infeasible. Given a node-to-subgraph assignment vector $\mathbf{x}$ and edge weight vector $\mathbf{w}$ there exists a feasible edge-to-subgraph assignment if and only if $G(\mathbf{x}, \mathbf{w}) \leq b$, i.e., the weight assigned to any subgraph does not exceed b. Thus the probabilistic edge-partition problem (2–1)–(2–4) seeks a minimum cost node-to-subgraph assignment such that the probability that there will be a feasible edge-to-subgraph assignment when the edge weights are realized is sufficiently high.

To build a scenario approximation of the probabilistic edge-partition problem (2–1)–(2–4), we generate an i.i.d. sample of $\tilde{\mathbf{w}}$ denoted by $\{\mathbf{w}^q\}_{q\in Q}$ (we call each realization a *scenario*). The scenario approximation is then:

$$\text{Minimize} \sum_{i\in N} \sum_{k\in K} x_{ik} \tag{2–10}$$

$$\text{subject to} \sum_{i\in N} x_{ik} \leq r \quad \forall k \in K \tag{2–11}$$

$$x_{ik} \in \{0,1\} \quad \forall i \in N,\ k \in K \tag{2–12}$$

$$G(\mathbf{x}, \mathbf{w}^q) \leq b \quad \forall q \in Q, \tag{2–13}$$

where the probabilistic constraint is replaced by the deterministic requirement that there must be a feasible edge-to-subgraph assignment for each scenario. As mentioned before we refer to the above problem as the stochastic edge-partition problem. The following result, which follows from the general results in Luedtke and Ahmed (2008), provides justification for considering the scenario approximation.

Proposition 1. *Let a desired confidence level $\delta \in (0,1)$ be given. If the sample size $|Q|$ satisfies*

$$|Q| \geq \frac{1}{\epsilon}\left[|N||K| \ln 2 - \ln \delta\right] \tag{2–14}$$

then any feasible solution to the stochastic edge-partition problem (2–10)–(2–13) is feasible to the probabilistic edge-partition problem (2–1)–(2–4) with probability at least $1-\delta$.

Proof. Let X denote the set of solutions satisfying the deterministic constraints (2–2) and (2–3), let X^ϵ denote the set of feasible solutions to the probabilistic edge-partition problem (satisfying (2–2)–(2–4)), and let X^Q denote the set of feasible solutions to the stochastic edge-partition problem corresponding to a sample Q (satisfying (2–11)–(2–13)). We want to bound $|Q|$ such that $\Pr\{X^Q \subseteq X^\epsilon\} \geq 1-\delta$.

Consider a solution $\mathbf{x} \in X \setminus X^\epsilon$, i.e., $\Pr\{G(\mathbf{x}, \tilde{\mathbf{w}}) \leq b\} < 1-\epsilon$. Then $\mathbf{x} \in X^Q$ if and only if $G(\mathbf{x}, \mathbf{w}^q) \leq b$ for all $q \in Q$. Since the $\mathbf{w}^q$ for $q \in Q$ are i.i.d. it follows that $\Pr\{\mathbf{x} \in X^Q\} \leq (1-\epsilon)^{|Q|}$. Now

$$\begin{aligned} \Pr\{X^Q \not\subseteq X^\epsilon\} &= \Pr\{\exists\, \mathbf{x} \in X^Q \text{ s.t. } \Pr\{G(\mathbf{x}, \tilde{\mathbf{w}}) \leq b\} < 1-\epsilon\} \\ &\leq \textstyle\sum_{\mathbf{x} \in X \setminus X^\epsilon} \Pr\{\mathbf{x} \in X^Q\} \leq |X \setminus X^\epsilon|(1-\epsilon)^{|Q|} \leq |X|(1-\epsilon)^{|Q|}. \end{aligned}$$

Thus $\Pr\{X^Q \subseteq X^\epsilon\} \geq 1 - |X|(1-\epsilon)^{|Q|}$. To guarantee that $\Pr\{X^Q \subseteq X^\epsilon\} \geq 1-\delta$ we need $|X|(1-\epsilon)^{|Q|} \leq \delta$ or equivalently

$$|Q| \geq \left[\ln |X| - \ln \delta\right] \Big/ \ln\left(\frac{1}{1-\epsilon}\right).$$

The claimed bound then follows by noting that $|X| \leq 2^{|N||K|}$ and $\ln(1/(1-\epsilon)) \geq \epsilon$. □

The above result suggests that we can obtain feasible solutions to the probabilistically constrained edge-partition problem by solving the stochastic edge-partition problem with a "not too large" number of scenarios. Key to this sampling-based approach is the ability to efficiently solve stochastic edge-partition instances having a modest number of scenarios, which is the motivation of this chapter.

Next we describe an *extensive form* model of the stochastic edge-partition problem. Let E^q be the set of edges with non-zero weights under scenario q. We define binary decision variables $y^q_{ijk} = 1$ if edge (i,j) is assigned to subgraph k in scenario q and 0 otherwise, $\forall q \in Q,\ (i,j) \in E^q$, and $k \in K$. The stochastic edge-partition problem can then be formulated as follows:

$$\text{Minimize} \sum_{i \in N} \sum_{k \in K} x_{ik} \tag{2–15}$$

$$\text{subject to} \sum_{k \in K} y^q_{ijk} = 1 \quad \forall q \in Q,\ (i,j) \in E^q \tag{2–16}$$

$$\sum_{i \in N} x_{ik} \leq r \quad \forall k \in K \tag{2–17}$$

$$\sum_{(i,j) \in E^q} w^q_{ij} y^q_{ijk} \leq b \quad \forall q \in Q,\ k \in K \tag{2–18}$$

$$y^q_{ijk} \leq x_{ik},\ y^q_{ijk} \leq x_{jk} \quad \forall q \in Q,\ (i,j) \in E^q,\ k \in K \tag{2–19}$$

$$x_{ik} \in \{0,1\} \quad \forall i \in N,\ k \in K \tag{2–20}$$

$$y^q_{ijk} \in \{0,1\} \quad \forall q \in Q,\ (i,j) \in E^q,\ k \in K. \tag{2–21}$$

Observe that if one were to solve the above extensive form problem given by (2–15)–(2–21), integrality restrictions need only be imposed on the y-variables, which would in turn enforce the integrality of the x-variables at optimality. Note also that given a fixed set of x-values, this problem decomposes into $|Q|$ separable integer programs, where the subproblem corresponding to scenario $q \in Q$ is given by:

$$S^q(x) = \text{Maximize } 0 \tag{2–22}$$

subject to (2–16), (2–18), (2–19), and (2–21). (2–23)

Under the foregoing model, it is useful to define $v_{ijk} = \min\{x_{ik}, x_{jk}\}$ as a part of the first-stage decision variables, $\forall (i,j) \in E,\ k \in K$. The presence of these variables allow us to formulate stronger cutting planes than would be possible with just x-variables (see also Smith et al. (2004)). Assuming that $\cup_{q \in Q} E^q = E$, the extensive form problem is now equivalent to:

$$\text{Minimize} \quad \sum_{i \in N} \sum_{k \in K} x_{ik} \tag{2–24}$$

$$\text{subject to} \quad \sum_{i \in N} x_{ik} \leq r \quad \forall k \in K \tag{2–25}$$

$$v_{ijk} \leq x_{ik},\ v_{ijk} \leq x_{jk} \quad \forall (i,j) \in E,\ k \in K \tag{2–26}$$

$$\sum_{k \in K} v_{ijk} \geq 1 \quad \forall (i,j) \in E \tag{2–27}$$

$$x_{ik} \in \{0,1\} \quad \forall i \in N,\ k \in K \tag{2–28}$$

$$F^q(\mathbf{v}) \leq b \qquad \forall q \in Q, \tag{2–29}$$

where

$$F^q(\mathbf{v}) = \quad \text{Minimize} \quad \max_{k \in K} \{ \sum_{(i,j) \in E^q} w_{ij}^q y_{ijk}^q \} \tag{2–30}$$

$$\text{subject to} \quad \sum_{k \in K} y_{ijk}^q = 1 \quad \forall (i,j) \in E^q \tag{2–31}$$

$$y_{ijk}^q \leq v_{ijk} \quad \forall (i,j) \in E^q,\ k \in K \tag{2–32}$$

$$y_{ijk}^q \in \{0,1\} \quad \forall (i,j) \in E^q,\ k \in K. \tag{2–33}$$

The valid inequalities (2–27) require that for each edge $(i,j) \in E$, both i and j must be assigned to some common subgraph, and are useful in improving the computational efficacy of the decomposition algorithm that we propose. Note that an optimal solution exists in which $v_{ijk} = \min\{x_{ik}, x_{jk}\}\ \forall (i,j) \in E,\ k \in K$, without enforcing integrality restrictions or lower bounds on the v-variables.

There can exist up to $|K|! - 1$ alternative optimal solutions to this problem by simply reindexing the subgraph indices. These symmetric solutions are known to impede the performance of branch-and-bound algorithms (Sherali and Smith, 2001; Sherali et al., 2000). To reduce model symmetry we can rewrite the cardinality constraints (2–25) (or (2–17) for the extensive form problem) by using the following inequalities:

$$r \geq \sum_{i \in N} x_{i1} \geq \sum_{i \in N} x_{i2} \geq \cdots \geq \sum_{i \in N} x_{i|K|}. \tag{2–34}$$

For a scenario q and a given vector $\hat{\mathbf{v}}$, the problem (2–30)–(2–33) is essentially an identical parallel machine scheduling problem to minimize makespan ($P/\ /C_{max}$) (with some assignment restrictions). In particular, there would be $|K|$ machines and $|E^q|$ jobs, whose processing times are given by $w^q_{ij},\ \forall (i,j) \in E^q$. Each job must be assigned to exactly one machine, and the v-variables impose some restrictions on the assignments. The integer programming scheme developed in Smith (2004) is tailored for a similar problem in which the (weighted) number of demands that cannot be placed on one of these subgraphs is minimized (i.e., minimum weighted number of tardy jobs). This is not equivalent to solving a minimum makespan problem; however, the optimal solution of $F^q(\hat{\mathbf{v}})$ is no more than b if and only if the minimum number of tardy jobs is equal to 0. If a positive lower bound to the problem of minimizing the number of tardy jobs is established, one can terminate the subproblem algorithm and conclude infeasibility.

We now present a cutting plane algorithm for solving (2–24)–(2–29). The scheme relaxes constraints (2–29) and adds cutting planes as necessary to enforce feasibility to the subproblems. Let us call the problem (2–24)–(2–28) the master problem (MP).

1. Solve MP. If MP is infeasible then STOP; the problem is infeasible. Otherwise let $\hat{\mathbf{v}}$ be an optimal solution of MP.

2. For $q \in Q$, compute $F^q(\hat{\mathbf{v}})$. If $F^q(\hat{\mathbf{v}}) \leq b$ for all q, then STOP; the current solution is optimal. Otherwise, continue to Step 3.

3. Update MP by adding a cutting plane of the form (2–37) as presented in Remark 1, and return to Step 1.

After a finite number of steps, the cutting plane algorithm terminates with an optimal solution or detects infeasibility.

Remark 1. *Suppose $F^{\hat{q}}(\hat{\mathbf{v}}) > b$ for some scenario $\hat{q}$ and a solution vector $\hat{\mathbf{v}}$ to MP. Let $L^{\hat{q}}$ be a global lower bound on $F^{\hat{q}}(v)$, i.e., $L^{\hat{q}} \leq F^{\hat{q}}(v)$ for all v. Also define $I(\hat{\mathbf{v}}) = \{(ijk) \ : \ \hat{v}_{ijk} = 1\}$ and $O(\hat{\mathbf{v}}) = \{(ijk) \ : \ \hat{v}_{ijk} = 0\}$. The integer optimality cut proposed by Laporte and Louveaux (1993) for this class of problems is given by*

$$(F^{\hat{q}}(\hat{\mathbf{v}}) - L^{\hat{q}}) \left[\sum_{(ijk)\in I(\hat{\mathbf{v}})} v_{ijk} - \sum_{(ijk)\in O(\hat{\mathbf{v}})} v_{ijk} \right] \leq b + (F^{\hat{q}}(\hat{\mathbf{v}}) - L^{\hat{q}})(|I(\hat{\mathbf{v}})| - 1) - L^{\hat{q}}. \quad (2\text{–}35)$$

Since $L^{\hat{q}} \leq b$ for any feasible instance, and since $F^{\hat{q}}(\hat{\mathbf{v}}) > b$ by assumption, we can apply Chvátal rounding to (2–35) by dividing both sides by $(F^{\hat{q}}(\hat{\mathbf{v}}) - L^{\hat{q}})$ and rounding down to obtain

$$\sum_{(ijk)\in O(\hat{\mathbf{v}})} v_{ijk} + \sum_{(ijk)\in I(\hat{\mathbf{v}})} (1 - v_{ijk}) \geq 1. \quad (2\text{–}36)$$

However, the following inequality is also a valid cutting plane that dominates (2–36):

$$\sum_{(ijk)\in O(\hat{\mathbf{v}})} v_{ijk} \geq 1. \quad (2\text{–}37)$$

To see that (2–37) is valid, consider a solution v' that does not satisfy the above inequality, i.e., $v'_{ijk} = 0$ for all $(ijk) \in O(\hat{\mathbf{v}})$. Therefore, $v'_{ijk} \leq \hat{v}_{ijk}$ for all (ijk). Then $F^q(v') \geq F^q(\hat{\mathbf{v}}) > b$, and v' is not feasible. Inequality (2–37) dominates (2–36) since the left-hand-side of (2–37) is not more than that of (2–36), and the right-hand-sides are both equal to 1. Thus, (2–37) serves as a cutting plane that can be used in Step 3 of the above algorithm.

Another reformulation of our subproblem might admit stronger cutting planes than the ones of the form (2–37). In the parlance of machine scheduling, instead of trying to minimize the maximum makespan, we may wish to minimize the total sum of tardiness. Let c_k, $\forall k \in K$ be a nonnegative variable that denotes the amount of capacity deficit in subgraph k. Then, the problem of minimizing the total capacity deficit can be formulated

as problem $\text{MT}^q(\mathbf{v})$ below:

$$\mathbf{MT}^q(\mathbf{v}): \; T^q(\mathbf{v}) = \text{Minimize} \sum_{k \in K} c_k \tag{2–38}$$

$$\text{subject to} \sum_{k \in K} y^q_{ijk} = 1 \quad \forall (i,j) \in E^q \tag{2–39}$$

$$y^q_{ijk} \leq v_{ijk} \quad \forall (i,j) \in E^q, \; k \in K \tag{2–40}$$

$$c_k \geq \sum_{(i,j) \in E^q} w^q_{ij} y^q_{ijk} - b \quad \forall k \in K \tag{2–41}$$

$$c_k \geq 0 \;\; \forall k \in K \tag{2–42}$$

$$y^q_{ijk} \in \{0,1\} \;\; \forall (i,j) \in E^q, \; k \in K. \tag{2–43}$$

Clearly, $F^q(\mathbf{v}) \leq b$ if and only if $T^q(\mathbf{v}) = 0$, and so we can replace master problem constraints (2–29) with the restrictions that $T^q(\mathbf{v}) = 0$ for all scenarios $q \in Q$. If subproblems $T^q(\mathbf{v})$ are used in lieu of $F^q(\mathbf{v})$, we would obtain (2–36) (directly, this time) from Laporte and Louveaux's integer feasibility cut. However, we can state a stronger cutting plane for a solution vector $\hat{\mathbf{v}}$ having $T^{\hat{q}}(\hat{\mathbf{v}}) > 0$ for some scenario $\hat{q}$, by requiring that the total amount of additional capacity that must be allocated to the collection of subgraphs is at least $T^{\hat{q}}(\hat{\mathbf{v}})$. This inequality is formally stated in the following proposition.

Proposition 2. *Suppose for some solution vector $\hat{\mathbf{v}}$ and for some scenario $\hat{q} \in Q$, we obtain a lower bound $LB^{\hat{q}}(\hat{\mathbf{v}}) > 0$ for $T^{\hat{q}}(\hat{\mathbf{v}})$. Then the following inequality is a valid cutting plane for problem MP, and is at least as strong as (2–37):*

$$\sum_{(ijk) \in O(\hat{\mathbf{v}})} \min\{w^{\hat{q}}_{ij}, LB^{\hat{q}}(\hat{\mathbf{v}})\} v_{ijk} \geq LB^{\hat{q}}(\hat{\mathbf{v}}). \tag{2–44}$$

Proof. Suppose by contradiction that there exists a binary vector $\mathbf{v}^*$ such that $T^{\hat{q}}(\mathbf{v}^*) = 0$, but $\sum_{(ijk) \in O(\hat{\mathbf{v}})} w^{\hat{q}}_{ij} v^*_{ijk} < LB^{\hat{q}}(\hat{\mathbf{v}})$. Then there exists a solution $(\mathbf{y}^*, \mathbf{c}^*)$ to $\text{MT}^{\hat{q}}(\mathbf{v}^*)$ having $c^*_k = 0 \; \forall k \in K$. We will show that the existence of such a $\mathbf{v}^*$ contradicts the assumption that $LB^{\hat{q}}(\hat{\mathbf{v}})$ is a valid lower bound on $T^{\hat{q}}(\hat{\mathbf{v}})$. We now build a solution $(\hat{\mathbf{y}}, \hat{\mathbf{c}})$ to $\text{MT}^{\hat{q}}(\hat{\mathbf{v}})$. First, we construct $\hat{\mathbf{y}}$ as follows:

1. For $(i,j) \in E^{\hat{q}}$, if $y^*_{ijk} = 1$ and $\hat{v}_{ijk} = 1$, then set $\hat{y}_{ijk} = 1$ as well.

2. For $(i,j) \in E^{\hat{q}}$, if $y^*_{ijk} = 1$ and $\hat{v}_{ijk} = 0$, then set $\hat{y}_{ij\hat{k}} = 1$ for any $\hat{k} \in K$ for which $(ij\hat{k}) \in I(\hat{\mathbf{v}})$. (Note that $(ijk) \in O(\hat{\mathbf{v}})$ since $\hat{v}_{ijk} = 0$.)

3. Set all other $\hat{y}_{ijk} = 0$.

In other words, $\hat{\mathbf{y}}$ is constructed in two phases. In the first phase, we ensure that if edge (i,j) was assigned to subgraph k in solution y^*, then (i,j) is assigned to k in $\hat{y}$ as well, unless $\hat{v}_{ijk} = 0$ (prohibiting this assignment). In the second phase, if $y^*_{ijk} = 1$ but $\hat{v}_{ijk} = 0$, then we assign (i,j) to any $\hat{k}$ such that $\hat{v}_{ij\hat{k}} = 1$. Note that this assignment results in a solution feasible to (2–39), (2–40), and (2–43). Next, let us construct $\hat{\mathbf{c}}$. Observe that in the first phase of assigning edges to subgraphs based on $(ijk) \in I(\hat{\mathbf{v}})$ for which $y^*_{ijk} = 1$, no subgraph capacities are violated since $c^*_k = 0$, $\forall k \in K$, and so we initialize $\hat{c}_k = 0$, $\forall k \in K$. In the second phase, we guarantee feasibility to (2–41) (and maintain feasibility to (2–42)) by increasing $\hat{c}_{\hat{k}}$ by $w^{\hat{q}}_{ij}$. Thus $(\hat{\mathbf{y}}, \hat{\mathbf{c}})$ is a feasible solution to $\mathrm{MT}^q(\hat{\mathbf{v}})$.

At the end of the second phase of assignments, we have $\sum_{k\in K} \hat{c}_k = \sum_{(ijk)\in O(\hat{\mathbf{v}})} w^{\hat{q}}_{ij} v^*_{ijk}$, since $\sum_{k\in K} \hat{c}_k$ is increased by $w^{\hat{q}}_{ij}$ only when both $v^*_{ijk} = 1$ and $(ijk) \in O(\hat{\mathbf{v}})$. However, by assumption, we have that $\sum_{(ijk)\in O(\hat{\mathbf{v}})} w^{\hat{q}}_{ij} v^*_{ijk} < LB^{\hat{q}}(\hat{\mathbf{v}})$. Since $\sum_{k\in K} \hat{c}_k = \sum_{(ijk)\in O(\hat{\mathbf{v}})} w^{\hat{q}}_{ij} v^*_{ijk}$, we have that $\sum_{k\in K} \hat{c}_k < LB^{\hat{q}}(\hat{\mathbf{v}})$, which contradicts the fact that $LB^{\hat{q}}(\hat{\mathbf{v}}) \le T^{\hat{q}}(\hat{\mathbf{v}})$. Therefore, all feasible solutions must obey the inequality

$$\sum_{(ijk)\in O(\hat{\mathbf{v}})} w^{\hat{q}}_{ij} v_{ijk} \ge LB^{\hat{q}}(\hat{\mathbf{v}}),$$

from which (2–44) is readily derived. Finally, by dividing both sides of (2–44) by $LB^{\hat{q}}(\hat{\mathbf{v}})$, we see that (2–44) implies (2–37). □

Remark 2. *In cutting plane implementations based on (2–37), once any scenario $\hat{q}$ is found such that the current $\hat{\mathbf{v}}$ vector is proven to be infeasible with respect to scenario $\hat{q}$, a cutting plane is generated and the master problem is re-solved. No further scenarios are tested, since an identical cut would be generated for each infeasible scenario. However, a cutting plane implementation based on problem (2–38)–(2–43) above with cutting planes*

(2–44) *might benefit from deriving multiple cuts for each infeasible scenario, since these cuts could be distinct.*

Remark 3. *Smith et al. (2004) explore the inclusion of "warming constraints" in the master problem, which enforce simple necessary conditions for feasibility to SONET problems. Denote the degree of node* $i \in N$ *by* $\deg(i)$*, and the set of nodes adjacent to* i *by* $A(i)$*. Lee et al. (2000b) show that node* i *must be assigned to at least* $\left\lceil \frac{\deg(i)}{r-1} \right\rceil$ *subgraphs, since otherwise, more than* r *nodes would be assigned to some subgraph. Similarly, for scenario* $q \in Q$*, the total weight associated with node* $i \in N$ *is given by* $\sum_{j \in A(i)} w_{ij}^q$*. Since the total weight that can be assigned to a subgraph is limited by* b*,* $\left\lceil \frac{\sum_{j \in A(i)} w_{ij}^q}{b} \right\rceil$ *is a lower bound on the number of copies of node* i*. We can then compute*

$$\ell_i = \max \left\{ \left\lceil \frac{\deg(i)}{r-1} \right\rceil, \max_{q \in Q} \left\lceil \frac{\sum_{j \in A(i)} w_{ij}^q}{b} \right\rceil \right\}, \tag{2–45}$$

and impose the following valid inequalities in the master problem:

$$\sum_{k \in K} x_{ik} \geq \ell_i \quad \forall i \in N. \tag{2–46}$$

Let $\hat{\imath}$ *denote a node having the largest lower bound, so that* $\ell_{\hat{\imath}} \geq \ell_i \ \forall i \in N$*. Node* $\hat{\imath}$ *can be assigned arbitrarily to subgraphs* $1, \ldots, \ell_{\hat{\imath}}$*, and we fix* $x_{\hat{\imath}1} = x_{\hat{\imath}2} = \cdots = x_{\hat{\imath}\ell_{\hat{\imath}}} = 1$ *accordingly. Note that the symmetry-breaking constraints (2–34) need to be adjusted so that they are enforced separately for subgraphs* $1, \ldots, \ell_{\hat{\imath}}$*, and* $\ell_{\hat{\imath}}+1, \ldots, |K|$*. Sherali et al. (2000) show computationally that such a variable-fixing scheme improves solvability of problem instances.*

Smith et al. (2004) note that a node i *cannot be assigned to a subgraph* k *in an optimal solution unless an adjacent node is also assigned to the same subgraph. Therefore, we also include the following constraints in MP:*

$$x_{ik} \leq \sum_{j \in A(i)} x_{jk} \quad \forall i \in N, \ k \in K. \tag{2–47}$$

Smith (2005) describes valid inequalities that can be derived by analyzing the topology of the graph. First, consider an edge $(i,j) \in E$ *such that* $\ell_i = \ell_j = 1$. *Let* $A(i,j) = A(i) \cup A(j) - \{i,j\}$ *denote the set of distinct nodes that are adjacent to* i *or* j. *If* $|A(i,j)| \geq r-1$, *then* i *or* j *must be assigned to at least two subgraphs. Similarly, we define* $W^q(i,j) = \sum_{k \in A(i,j)}(w^q_{ik} + w^q_{jk}) + w^q_{ij}$, *and note that if* $W^q(i,j) > b$ *for some* $q \in Q$, *then we cannot feasibly assign nodes* i *and* j *to a single subgraph. If* $A(i,j) \geq r-1$ *or* $W^q(i,j) > b$, *then we state the following valid inequality:*

$$\sum_{k \in K} x_{ik} + \sum_{k \in K} x_{jk} \geq 3. \tag{2–48}$$

Second, for each edge $(i,j) \in E$, *suppose* $deg(i) \geq r$, $deg(j) < r$, *and* $|A(i,j)| > 2r-3$. *Smith (2005) shows that nodes* i *and* j *collectively need to be assigned to at least four subgraphs, which we state as:*

$$\sum_{k \in K} x_{ik} + \sum_{k \in K} x_{jk} \geq 4. \tag{2–49}$$

2.3 A Hybrid IP/CP Approach

The cutting plane algorithms presented in Section 2.2 are preferable to solving stochastic edge-partition instances by the extensive form problem given by (2–15)–(2–21), as we show in Section 2.4. However, the two-stage cutting plane algorithms still suffer from several computational difficulties. First, the master problem, MP, contains $|N||K|$ binary variables, $|E||K|$ continuous variables, and $O(|E||K|)$ constraints, which results in large integer programs. Second, the linear programming relaxation of MP is quite weak for many problem instances. Furthermore, the lower bound improves slowly as cuts of the type (2–37) or (2–44) are added to MP in each iteration. The main reason for this slow convergence is the existence of symmetry in MP. Inequalities (2–34) reduce, but do not completely eliminate, symmetric solutions in MP. Therefore, when a solution of MP is found to be infeasible to a subproblem, MP often simply switches to a symmetric solution

having the same objective function value. On the other hand, stronger anti-symmetry constraints tend to make MP very difficult to solve.

In this section we develop a new decomposition framework to remedy these difficulties. We combat symmetry due to reshuffling of subgraphs by representing subgraphs as *configurations*. A configuration c is identified by a subgraph node set N_c (we allow $N_c = \emptyset$) and a positive integer α_c, which gives the number of subgraphs having node set N_c. A solution is represented by a *configuration multiset* C whose elements are pairs (N_c, α_c). We eliminate symmetry by ensuring that no isomorphic configuration multisets (i.e., those that are identical after reindexing configuration indices) are encountered in our search.

A configuration multiset C satisfies the following necessary feasibility conditions.

F1: $\sum_{c \in C} \alpha_c = |K|$ (partitions E into $|K|$ subgraphs)

F2: $|N_c| \leq r, \ \forall c \in C$ (no subgraph contains more than r nodes)

F3: $\forall (i,j) \in E, \ \exists c \in C$ such that $i \in N_c, \ j \in N_c$ (for each edge (i,j), there is at least one subgraph to which (i,j) can be assigned)

A multiset C that satisfies F1, F2, and F3 represents a feasible solution if all edges can be partitioned on the set of subgraphs corresponding to C without violating the weight restrictions for any scenario. Note that the number of distinct configurations in C, which we denote by $|C|$, is dynamically determined in our algorithm.

We now provide an overview of our three-stage hybrid algorithm.

1. The first-stage problem determines (via optimal solution of a mixed-integer program) the number of times we assign each node to the configurations in C. For instance, in the example given in Figure 2-1a, we could specify that we must use two copies of nodes 4 and 5, and one copy of the other nodes.

2. In the second stage, we seek a multiset C that uses exactly the number of node assignments specified in the first phase and satisfies F1, F2, and F3. In the example mentioned above, a multiset C having configurations $\{1, 2, 4\}$, $\{3, 4, 5\}$, and $\{5, 6\}$ (each with multiplicity one) could be generated based on the first-stage solution.

3. Finally, in the third stage, we determine whether C is feasible. If C is feasible then we stop with an optimal solution. Else, we return to the second stage, and generate a different multiset meeting the stated criteria. If no such multiset exists, a cut is added to the first-stage problem, which is then re-solved. For the example given above, the multiset yields a feasible solution (see Figure 2-1b).

2.3.1 First-Stage Problem

For all $i \in N$, let z_i be an integer variable that represents the number of copies of node i to be used in forming configurations. We say that an $|N|$-dimensional vector $\mathbf{z}$ *induces* a multiset C if C contains exactly z_i copies of node i, $\forall i \in N$. The first-stage problem can succinctly be written as:

$$\text{Minimize} \quad \sum_{i \in N} z_i \tag{2–50}$$

$$\text{subject to} \quad \mathbf{z} \text{ induces a feasible multiset} \tag{2–51}$$

$$\ell_i \leq z_i \leq |K| \quad \forall i \in N \tag{2–52}$$

$$z_i \text{ integer}, \tag{2–53}$$

where ℓ_i is a lower bound on the number of copies required for node i, as given in (2–45). To formulate the first-stage problem as an integer program, we rewrite (2–51) as an exponential set of linear inequalities by considering the z-vectors that violate it. We first need to introduce auxiliary binary variables t_{ik}, $\forall i \in N$, $k = \ell_i, \ldots, |K|$, so that $t_{ik} = 1$ if $z_i = k$. Then, given a vector $\hat{\mathbf{z}}$ that does not induce a feasible multiset, we note that no $\bar{\mathbf{z}}$ such that $\bar{z}_i \leq \hat{z}_i$, $\forall i \in N$, induces a feasible multiset. Hence, at least one component of $\hat{\mathbf{z}}$ must be increased, and so

$$\sum_{i \in N} \sum_{k=\hat{z}_i+1}^{|K|} t_{ik} \geq 1 \tag{2–54}$$

is a valid inequality. Our first-stage problem can now be expressed as the following integer program:

$$\text{Minimize} \quad \sum_{i \in N} z_i \tag{2–55}$$

$$\text{subject to} \quad z_i = \sum_{k=\ell_i}^{|K|} k t_{ik} \quad \forall i \in N \tag{2–56}$$

$$\sum_{k=\ell_i}^{|K|} t_{ik} = 1 \quad \forall i \in N \tag{2–57}$$

$$\sum_{i \in N} \sum_{k=\hat{z}_i+1}^{|K|} t_{ik} \geq 1 \quad \forall \hat{\mathbf{z}} \in \mathcal{Z} \tag{2–58}$$

$$t_{ik} \text{ binary} \quad \forall i \in N,\ k = \ell_i, \ldots, |K|, \tag{2–59}$$

where $\mathcal{Z}$ is the set of all z-vectors that do not induce a feasible multiset. (The z-variables are in fact unnecessary in this formulation, but we keep them for ease of exposition.) In our algorithm we relax constraints (2–58) in the first-stage problem, and add them in a cutting plane fashion. In every iteration we solve the first-stage problem to find $\hat{\mathbf{z}}$, and solve the second- and third-stage problems to seek a feasible multiset induced by $\hat{\mathbf{z}}$. If a feasible multiset is found, then $\hat{\mathbf{z}}$ induces an optimal solution and we stop. Otherwise, we add a cut of type (2–58) and re-solve the first-stage problem.

2.3.2 Second-Stage Problem

Our second-stage problem seeks a multiset induced by $\hat{\mathbf{z}}$ that satisfies F1, F2, and F3, using a constraint programming search. Given a set of constraints, a set of variables, and the domain of each variable (i.e., the set of values that each variable can take), constraint programming seeks a value assignment to each variable that satisfies all constraints. Constraints are propagated to reduce variable domains, which in turn trigger new constraint propagations. When no more domain reductions are possible, the algorithm searches for a solution by fixing a variable to a value in its domain, then recursively propagating constraints and reducing variable domains. If the domain of a variable becomes empty during constraint propagation, then the algorithm backtracks. We refer the reader to Smith (1995), Lustig and Puget (2001), and Rossi et al. (2006) for a thorough discussion of constraint programming techniques.

2.3.2.1 Foundations

In our second-stage algorithm, a *solution* corresponds to a multiset C induced by $\hat{\mathbf{z}}$ that meets conditions F1, F2, and F3. In a solution each node i has a corresponding $|C|$-dimensional *distribution vector* β^i, which represents the number of copies of node i to be allocated to each existing configuration in C. Note that β^i_c cannot exceed α_c, and that $\sum_{c \in C} \beta^i_c = \hat{z}_i$. The *domain* of a node $i \in N$ is the set of possible β^i-vectors that i can take. We say that a node i is *processed* if we have selected its distribution vector β^i. A *partial multiset* is constructed by processing a subset of the nodes in N.

For instance, consider a five-node graph, and let the z-vector obtained by the first-stage problem be $\hat{\mathbf{z}} = (2, 3, 1, 4, 3)$. Suppose that nodes 1, 2, and 3 have been processed, and the following partial multiset with $|C| = 3$ has been obtained:

- $N_1 = \emptyset,\ \alpha_1 = 5$,
- $N_2 = \{1, 2\},\ \alpha_2 = 2$, and
- $N_3 = \{2, 3\},\ \alpha_3 = 1$.

Suppose that we process node 4 by choosing its distribution vector as $\hat{\beta}^4 = (2, 1, 1)$. Adding node 4 to two of the five copies of N_1 creates a new configuration N'_1 whose node set consists only of node 4 (with multiplicity two) and reduces the multiplicity of N_1 by two. After similarly adding one copy of node 4 to N_2 and one copy of node 4 to N_3, we obtain the following partial multiset with $|C'| = 5$:

- $N_1 = \emptyset,\ \alpha_1 = 3$ (reduced α_1),
- $N'_1 = \{4\},\ \alpha'_1 = 2$ (generated from configuration 1 by adding node 4 to N_1),
- $N_2 = \{1, 2\},\ \alpha_2 = 1$ (reduced α_2),
- $N'_2 = \{1, 2, 4\},\ \alpha'_2 = 1$ (generated from configuration 2 by adding node 4 to N_2), and
- $N_3 = \{2, 3, 4\},\ \alpha_3 = 1$ (added node 4 to N_3).

In general, when we process node i by choosing a distribution vector β^i, we update the partial multiset C as follows. For each configuration $c \in C$ if $\beta^i_c = 0$, then no changes

are made to c (since no copies of node i are added to c). If $\beta_c^i = \alpha_c$, then we update configuration c by setting $N_c = N_c \cup \{i\}$. Finally, if $0 < \beta_c^i < \alpha_c$, then we create a new configuration c' having $N_{c'} = N_c \cup \{i\},\ \alpha_{c'} = \beta_c^i$, and update configuration c by setting $\alpha_c = \alpha_c - \beta_c^i$.

Remark 4. *Recall that the configurations in a partial multiset C can be ordered in $|C|!$ symmetric ways. Our algorithm avoids this symmetry by generating only one such ordering after processing a node. Furthermore, the configuration multisets that we compute by processing node i according to the β^i-vectors in its domain must be pairwise nonisomorphic, since the β^i-values in the domain of node i are distinct. Hence, we never encounter isomorphic configuration multisets in the second-stage search.*

2.3.2.2 Domain expansion

Processing a node modifies the current partial multiset, and therefore distribution vectors of the remaining unprocessed nodes need to be updated. Domains of nodes are reduced by constraint propagation as we describe in the next section, but must also be expanded as new configurations are generated. We describe the initialization and expansion of node domains below.

In the beginning of the second stage we initialize our multiset C with a single configuration having $N_1 = \emptyset$ and $\alpha_1 = |K|$. Each node can only be added to the lone configuration, and so the domain for node i is initially the single one-dimensional vector $\beta^i = (\hat{z}_i)$. Our algorithm next processes some node $i \in N$ and updates the existing set of configurations: $N_1 = \emptyset,\ \alpha_1 = |K| - \hat{z}_i$ and $N_2 = \{i\},\ \alpha_2 = \hat{z}_i$. Next, the domains of all unprocessed nodes are updated to reflect the changes in C. For each unprocessed node j, we enumerate all possible ways of partitioning $\hat{z}_j$ copies into node sets N_1 and N_2. This logic is repeated at all future steps as well. For instance, in the example given above, suppose that $\hat{\beta}^5 = (2, 0, 1)$ was the only vector in the domain of node 5 before processing node 4. Since processing node 4 modifies the first configuration by reducing α_1 and generates a new configuration (N_1', α_1'), we expand the domain of node 5 by enumerating

all possible ways of assigning $\hat{\beta}_1^5 = 2$ copies of node 5 to configurations (N_1, α_1) and (N_1', α_1'). On the other hand, since $\hat{\beta}^5$ does not assign node 5 to the second configuration, the distribution vectors in the expanded domain do not add node 5 to (N_2, α_2) or (N_2', α_2'). Finally, since processing node 4 does not generate any new configurations from the third configuration, all distribution vectors in the expanded domain of node 5 assign a single copy of node 5 to (N_3, α_3). After processing node 4 and updating the configurations as described above, the domain of node 5 is expanded to:

$$\{(2,0,0,0,1),\ (1,1,0,0,1),\ (0,2,0,0,1)\}.$$

2.3.2.3 Constraint propagation

The constraints we impose in the second-stage problem limit the number of nodes in each configuration (F2) and require that each edge has both its end points in at least one configuration (F3). Condition F1 (requiring $|K|$ total configurations) is implicitly satisfied. We apply constraint propagation algorithms to remove distribution vectors inconsistent with F2 or F3 from the expanded node domains. Let $i \in N$ be the last processed node, and let $C_i \subseteq C$ represent the subset of configurations to which node i has been added. We only need to execute constraint propagation for configurations $c \in C_i$, since these are the only newly modified configurations.

To enforce F2, the propagation algorithm identifies all configurations to which r nodes have been assigned. For each such configuration c, we remove all distribution vectors β^j having $\beta_c^j > 0$ from the domains of all unprocessed nodes $j \in N$. To enforce F3, the propagation algorithm iterates over the domains of the unprocessed nodes j adjacent to i, and removes all distribution vectors that do not add at least one copy of j to any configuration in C_i. Otherwise, the configurations containing node i would be disjoint from those containing node j, which violates F3.

2.3.2.4 Forward checking

After all constraints are propagated, we first check whether the domain of any unprocessed node is empty; if so, then we backtrack. Else, we further analyze the current partial multiset before resuming the search with the next unprocessed node. This step identifies whether the current partial multiset can eventually yield a feasible multiset as early as possible to avoid performing unnecessary backtracking steps (van Beek, 2006).

We call one such test *implied node assignment analysis*. Suppose that we identify a processed node i such that $\hat{z}_i = 1$, and the configuration c to which i has been assigned. By condition F3 it follows that all unprocessed nodes j adjacent to i must also be assigned to configuration c. We use this analysis to augment partial configurations with implied node assignments, and then check whether any augmented configuration contains more than r nodes, and hence violates F2.

We also perform an *implied edge assignment analysis* by finding all edges that can only be assigned to a single configuration. For each $(i, j) \in E$, if both nodes i and j have been processed, then we check whether both i and j are in a single configuration c for which $\alpha_c = 1$. In this case edge (i, j) can only be assigned to configuration c. On the other hand if (without loss of generality) node i has been processed but node j has not yet been processed, and $\hat{z}_i = 1$, then edge (i, j) can only be assigned to the configuration to which i has been assigned. After finding all implied edge assignments, we check whether F3 is violated for any scenario.

Finally, we consider a *singleton node analysis*, in which we ensure that each node is adjacent to at least one other node in each configuration. For each processed node i, and for all configurations $c \in C_i$, we seek a node j adjacent to i so that either $j \in N_c$ (if j also has been processed), or $\beta_c^j > 0$ for some distribution vector in the domain of j (if j has not been processed). If no such j can be found for a configuration $c \in C_i$, then the current partial solution cannot lead to an optimal solution; node i can ultimately be removed

from configuration c without affecting feasibility conditions, leading to a reduction in the objective function value.

2.3.2.5 Node selection rule

The order in which variables are processed can significantly affect the performance of constraint programming algorithms (Lustig and Puget, 2001; Smith, 1995). Especially for infeasible second-stage problem instances, processing the "problematic" nodes first can quickly lead to the detection of infeasibility and can result in significant savings in computational time. We employ a dynamic node selection rule in which the order of nodes considered can vary in different sections of the search tree. In accordance with the "fail-first" principle widely used in constraint programming algorithms (Haralick and Elliott, 1980; van Beek, 2006), our node selection rule first picks an unprocessed node that

1. has the fewest number of distribution vectors in its domain,
2. has the fewest number of copies to be partitioned, and
3. has the largest number of unprocessed adjacent nodes,

breaking ties in the given order. In this manner, we can quickly enumerate all possible distribution vectors of a few key nodes, allowing constraint propagation to quickly reduce the size of the remaining search space.

2.3.2.6 Distribution vector ordering rule

Once the next node to be processed has been identified, all distribution vectors in its domain need to be tried one-by-one to see if any of them leads to a feasible multiset. For an infeasible second-stage problem instance, the order in which these vectors are instantiated does not matter, because all vectors must be enumerated before infeasibility can be concluded. However, for feasible problem instances it is important to find a vector that leads to a feasible multiset as soon as possible to curtail our search. Our ordering rule attempts to sort the distribution vectors in nonincreasing order of the likelihood that the vector leads to a feasible multiset. We calculate the feasibility likelihood score of a distribution vector β^i in the domain of an unprocessed node i with respect to a partial

multiset C as:

$$FL(i, C, \beta^i) = \sum_{c \in C} \beta^i_c |\{j \in N_c : (i, j) \in E\}|. \tag{2–60}$$

$FL(i, C, \beta^i)$ measures the total number of adjacent node pairs (i, j) that would be added across all configurations if β^i is selected to be the distribution vector for node i. Our vector ordering rule sorts vectors in the domain of the chosen node in nondecreasing order of their FL-scores. By allowing for a higher degree of flexibility in assigning edges, we increase the likelihood that a feasible partition of edges to subgraphs can be found.

2.3.3 Third-Stage Problem

Given a solution of the second-stage problem that consists of a configuration multiset C satisfying F1, F2, and F3, the third-stage problem must verify whether C is feasible. We first generate the set of subgraphs from the multiset $C = \{c_1, c_2, \ldots, c_{|C|}\}$ by assigning the nodes in N_{c_1} to the first α_{c_1} subgraphs, then assigning the nodes in N_{c_2} to the next α_{c_2} subgraphs, and so on. Since we have enforced $\sum_{c \in C} \alpha_c = |K|$, this transformation creates exactly $|K|$ subgraphs, some of which can be empty. Then we iterate over all subgraphs and set $v_{ijk} = 1$ if nodes i and j are in subgraph k, and $v_{ijk} = 0$ otherwise. We then use formulation (2–38)–(2–43) to solve the third-stage problem.

Note that this transformation re-introduces symmetry into the third-stage problem. However, the solution of the third-stage problems does not constitute a bottleneck in the algorithm, and symmetry-breaking constraints appended to the transformed subproblem will not impact the computational efficacy of the overall algorithm.

2.3.4 Infeasibility Analysis

If a z-vector is found not to induce a feasible multiset, we add a constraint to the first-stage problem so that the same z-vector is not generated in subsequent iterations. Constraints (2–58) state that the number of copies of some node must be increased, but they do not contain any information about *which* nodes need to be added. We observe that the progress of our second-stage algorithm can be analyzed to identify a "problematic" subset of nodes whose corresponding z-values cause infeasibility regardless

of other variable values. Given a vector $\hat{\mathbf{z}}$ for which no feasible multiset exists, if a node $i \in N$ has not been processed, or has not been identified as the reason of infeasibility in any step of the backtracking algorithm, then $\hat{\mathbf{z}}$ will not induce a feasible multiset for any value of $\hat{z}_i$. Let $P \subseteq N$ denote the set of nodes that have been processed, or whose domains have become empty due to constraint propagation in the second-stage algorithm, possibly during different backtracking steps. The following is a valid inequality:

$$\sum_{i \in P} \sum_{k=\hat{z}_i+1}^{|K|} t_{ik} \geq 1. \tag{2–61}$$

Constraints (2–61) clearly dominate (2–58) for any $P \subset N$, and get stronger as $|P|$ decreases. Based on this observation, we update our node selection rule by giving preference to selecting nodes that have already been added to P. Our revised node selection rule first picks a node that

0. has been added to P in a previous backtracking step,
1. has the fewest number of distribution vectors in its domain,
2. has the fewest number of copies to be partitioned, and
3. has the largest number of unprocessed adjacent nodes,

again breaking ties in the stated order.

2.3.5 Enhancements for the First-Stage Problem

Our computational studies revealed that the first-stage integer programming model solution represents the bottleneck operation of our algorithm. To decrease the computational time spent by the first-stage problem, we investigate several strategies.

2.3.5.1 Valid inequalities

The valid inequalities that we discuss in Remark 3 can be adapted to the first-stage problem to eliminate the z-vectors that violate the corresponding necessary feasibility conditions. In particular, constraints (2–46) translate to simple lower bounds (2–52) on the z-variables. Constraints (2–48), which are written for node pairs that satisfy the

conditions discussed in Remark 3, can be written as:

$$z_i + z_j \geq 3. \tag{2–62}$$

Similarly, each constraint of type (2–49) can be equivalently represented as following:

$$z_i + z_j \geq 4. \tag{2–63}$$

Smith (2005) discusses an additional valid inequality, which cannot be represented using the x-variables in our two-stage algorithm, but can be written in terms of the z- and t-variables in the first-stage problem of our hybrid algorithm. For nodes $i \in N$ and $j \in N$, if $(i,j) \notin E, deg(i) \leq r-1, deg(j) \leq r-1,\ |A(i,j)| \geq r-1$, and there exists a common neighbor $k \in N$ so that $k \in A(i),\ k \in A(j),\ deg(k) \geq r$, and if i, j, k have more than $2r-4$ distinct neighbors in total, then $z_i = 1,\ z_j = 1$ implies $z_k \geq 3$. This condition can be written as:

$$z_k \geq -1 + 2(t_{i1} + t_{j1}), \tag{2–64}$$

which reduces to $z_k \geq 3$ for $z_i = z_j = 1$, and is redundant otherwise.

2.3.5.2 Heuristic for obtaining an initial feasible solution

The existence of a good initial feasible solution can help improve the performance of the first-stage problem because it provides a good upper bound, and allows the solver to apply strategies such as reduced cost fixing. We first solve the first-stage model enhanced with valid inequalities (2–62)–(2–64) to obtain an initial solution $\hat{\mathbf{z}}$, and execute the second- and third-stage algorithms to seek a feasible multiset. If one is found, we terminate with an optimal solution. Otherwise, we investigate the set of processed nodes $\hat{P} \subseteq N$, and pick a node $\hat{\imath} \in \hat{P}$ having the fewest number of copies (breaking ties by picking a node having the largest degree). We then set $\hat{z}_{\hat{\imath}} = \hat{z}_{\hat{\imath}} + 1$ and re-invoke the second- and third-stage algorithms. This algorithm eventually finds a feasible multiset or concludes that the entire problem is infeasible after generating the solution $\hat{z}_i = |K|,\ \forall i \in$

N. We also generate a cut of type (2–61) for each $\hat{\mathbf{z}}$ generated before a feasible multiset is found, which we add to the first-stage problem to improve the lower bound.

2.3.5.3 Processing integer solutions

We can interrupt the branch-and-bound solution process of the first-stage problem each time the solver finds an integer solution $\hat{\mathbf{z}}$, and check whether $\hat{\mathbf{z}}$ induces a feasible multiset by solving the second- and third-stage problems. If a feasible multiset exists, we accept $\hat{\mathbf{z}}$ as the new incumbent and resume solving the first-stage problem. Otherwise, we reject $\hat{\mathbf{z}}$, generate a constraint of type (2–61), and again resume the solution process. The same idea is also applicable to the master problem (MP) of the two-stage algorithm discussed in Section 2.2.

In our tests, this approach turned out to be more effective than solving the first-stage problem to optimality in each iteration, adding a cut, and re-solving it. The reason is that the problem is solved using a single branch-and-bound tree, which we tighten by adding cuts as necessary on integral nodes, instead of repeatedly generating a branch-and-bound tree in each iteration. It also allows us to obtain good feasible solutions for problem instances that are too difficult to solve to optimality.

We note that this approach requires a minor modification to the second-stage algorithm. All constraint propagation (Section 2.3.2.3) and forward checking rules (Section 2.3.2.4) except for singleton node analysis are based on necessary conditions for feasibility of configurations, and therefore they are valid for any integral $\hat{\mathbf{z}}$. However, singleton node analysis is based on an optimality condition and hence can only be used if $\hat{\mathbf{z}}$ is a candidate optimal solution to the first-stage problem.

2.4 Computational Results

We implemented the algorithms discussed in the previous sections using CPLEX 11.1 running on a Windows XP PC with a 3.4 GHz CPU and 2 GB RAM. Our base set of test problem instances consists of 225 randomly generated problem instances for which the expected edge density of the graph (measured as $\frac{|E|}{|N|\times(|N|-1)}$) takes values 0.2, 0.3, and

0.4, the number of nodes ranges from 5 to 15, and the number of scenarios is between 1 (corresponding to the deterministic edge-partition problem) and 100. There is no practical limit on the number of subgraphs ($|K|$), but a limit needs to be specified to model the problem (see Goldschmidt et al. (2003); Sherali et al. (2000); Smith (2005)). Choosing $|K|$ too small may make the problem infeasible, and large values of $|K|$ increase difficulty of the problem. In our tests, we chose $|K|$ sufficiently large to yield a feasible edge partition in each problem instance. In generating instances we first picked a random subset of edges to have a positive weight, and then we assigned a weight uniformly distributed between 1 and 10 to each edge in each scenario. We generated five problem instances for each problem size, which is determined by the expected edge density, the number of nodes, and the number of scenarios. The data set names and details used in our experiments are given in Table 2-1.

Table 2-1. Descriptions of the problem instances used for comparing algorithms

Name	$\|N\|$	$\|K\|$	$\|Q\|$	r	b	Name	$\|N\|$	$\|K\|$	$\|Q\|$	r	b
5-1	5	5	1	4	20	12-1	12	10	1	5	50
5-30	5	5	30	4	20	12-30	12	10	30	5	50
5-100	5	5	100	4	20	12-100	12	10	100	5	50
8-1	8	7	1	4	35	15-1	15	10	1	8	70
8-30	8	7	30	4	35	15-30	15	10	30	8	70
8-100	8	7	100	4	35	15-100	15	10	100	8	70
10-1	10	8	1	5	40						
10-30	10	8	30	5	40						
10-100	10	8	100	5	40						

We used the default options of CPLEX for solving the extensive form problems. Preliminary computational experience on our two-stage algorithm indicated that the best implementation includes the valid inequalities (2–27), (2–46)–(2–49), and the symmetry-breaking constraints (2–34), and uses the model given by (2–38)–(2–43) for the subproblem, which is the formulation that minimizes the total tardiness. In our base setting for the three-stage algorithm, we used our heuristic to find an initial feasible solution, generated valid inequalities (2–62)–(2–64), and (similar to the two-stage algorithm) we used formulation (2–38)–(2–43) for the third-stage problem. We used callback functions of CPLEX to generate a single branch-and-bound tree for both two-stage and three-stage algorithms as discussed in Section 2.3.5. We imposed a half-hour

(1800 seconds) time limit past which we halted the execution of an algorithm in all our experiments.

Our first experiment compares the performance of the extensive form, two-stage, and three-stage algorithms. Table 2-2 summarizes the results of these three algorithms on low density graphs having expected edge density 0.2. For each problem size, we report the following statistics calculated over five random instances: (i) the number of problems solved to optimality ("Solved"), (ii) the average optimality gap obtained at the root node ("Root Gap"), (iii) the average final optimality gap for instances that could not be solved within the allowed time limit ("Final Gap"), (iv) the average amount of time spent by each algorithm on the instances that were solved to optimality ("Time"). Out of the 75 instances in this data set, CPLEX could solve the extensive form to optimality for 61 instances, while both two-stage and three-stage algorithms solved all 75 instances to optimality within a few seconds. The results reveal that the performance of the extensive form formulation deteriorates rapidly as the number of scenarios increases, but the effect of the number of scenarios is mitigated for the two-stage and three-stage algorithms. We observe that the average optimality gap obtained by the three-stage algorithm at the root node is 1.46%, which is significantly less than the initial gaps obtained using other approaches.

Table 2-2. Comparison of the algorithms on graphs having edge density = 0.2

	Extensive Form				Two-Stage				Three-Stage			
Name	Solved	Root Gap	Final Gap	Time	Solved	Root Gap	Final Gap	Time	Solved	Root Gap	Final Gap	Time
5-1	5	0.00%	-	0.1	5	5.00%	-	0.1	5	0.00%	-	0.1
5-30	5	18.33%	-	6.6	5	4.00%	-	0.2	5	0.00%	-	0.1
5-100	5	12.38%	-	5.4	5	11.00%	-	0.6	5	0.00%	-	0.3
8-1	5	25.90%	-	0.4	5	6.67%	-	0.1	5	0.00%	-	0.1
8-30	5	12.89%	-	4.0	5	3.64%	-	0.2	5	0.00%	-	0.1
8-100	5	37.61%	-	223.1	5	14.84%	-	1.2	5	0.00%	-	0.3
10-1	5	19.58%	-	0.5	5	17.80%	-	0.4	5	0.00%	-	0.1
10-30	5	57.01%	-	147.3	5	10.71%	-	0.8	5	0.00%	-	0.2
10-100	4	30.35%	7.14%	684.1	5	13.94%	-	2.0	5	0.00%	-	0.4
12-1	5	47.25%	-	8.1	5	24.66%	-	2.2	5	0.00%	-	0.1
12-30	4	55.09%	25.00%	507.3	5	17.99%	-	4.3	5	3.08%	-	0.3
12-100	2	62.21%	24.88%	713.1	5	36.28%	-	4.7	5	2.11%	-	0.8
15-1	5	31.85%	-	33.0	5	64.38%	-	16.4	5	4.56%	-	0.2
15-30	1	65.29%	21.65%	864.6	5	39.13%	-	27.1	5	7.29%	-	0.6
15-100	0	57.33%	28.47%	-	5	24.49%	-	20.4	5	4.86%	-	1.2

Tables 2-3 and 2-4 compare the three approaches on denser graphs having edge density 0.3 (medium density) and 0.4 (high density), respectively. We observe that performances of all three algorithms deteriorate as the edge density increases, which is not surprising due to the nature of the edge-partition problem. The number of instances

Table 2-3. Comparison of the algorithms on graphs having edge density = 0.3

	Extensive Form				Two-Stage				Three-Stage			
		Root	Final			Root	Final			Root	Final	
Name	Solved	Gap	Gap	Time	Solved	Gap	Gap	Time	Solved	Gap	Gap	Time
5-1	5	0.00%	-	0.1	5	2.86%	-	0.1	5	0.00%	-	0.1
5-30	5	25.76%	-	10.4	5	6.15%	-	0.5	5	0.00%	-	0.1
5-100	5	10.00%	-	3.1	5	10.77%	-	0.4	5	0.00%	-	0.2
8-1	5	31.30%	-	0.5	5	11.20%	-	0.1	5	0.00%	-	0.1
8-30	5	42.57%	-	18.0	5	7.48%	-	0.4	5	0.00%	-	0.2
8-100	4	39.37%	7.14%	110.0	5	16.19%	-	1.3	5	1.43%	-	0.3
10-1	5	32.42%	-	3.7	5	16.27%	-	0.6	5	1.18%	-	0.1
10-30	4	51.33%	21.05%	953.0	5	40.82%	-	8.2	5	5.83%	-	0.3
10-100	2	61.24%	29.05%	382.6	5	35.43%	-	302.7	5	8.89%	-	0.5
12-1	5	53.85%	-	312.0	5	39.49%	-	16.8	5	4.65%	-	0.2
12-30	0	63.41%	27.06%	-	5	46.98%	-	120.5	5	9.31%	-	0.8
12-100	0	84.24%	65.50%	-	4	42.78%	4.35%	89.0	5	11.99%	-	1.4
15-1	4	46.88%	11.54%	460.4	5	72.86%	-	250.5	5	12.93%	-	0.9
15-30	0	66.01%	42.41%	-	2	72.20%	16.02%	30.4	5	13.51%	-	3.5
15-100	0	80.76%	74.48%	-	0	53.05%	13.21%	-	5	16.31%	-	4.1

that can be solved by the extensive form decreases from 61 for low density graphs to 49 for medium density graphs, and finally to 36 for high-density graphs. The two-stage algorithm also exhibits a similar behavior; it can solve 75, 66, and 61 instances for low, medium, and high-density graphs, respectively. On the other hand, the three-stage algorithm is able to solve almost all instances, failing to solve two instances in the high-density 15-30 and 15-100 data sets to optimality within the allowed time limit. Table 2-4 clearly shows that the three-stage algorithm dominates the other approaches, and the two-stage algorithm provides better results than directly solving the extensive formulation. Our analysis of optimal solutions obtained for the problem instances shown in Tables 2-2–2-4 showed that the average objective function value for the deterministic (single-scenario) problem instances is 14.8. This value is smaller than the average objective function value for 30-scenario and 100-scenario instances (15.52 and 15.6, respectively). We also observe that several subgraphs can be empty in an optimal solution.

Table 2-4. Comparison of the algorithms on graphs having edge density = 0.4

	Extensive Form				Two-Stage				Three-Stage			
Name	Solved	Root Gap	Final Gap	Time	Solved	Root Gap	Final Gap	Time	Solved	Root Gap	Final Gap	Time
5-1	5	5.00%	-	0.1	5	0.00%	-	0.1	5	0.00%	-	0.1
5-30	5	24.67%	-	2.6	5	19.79%	-	0.3	5	0.00%	-	0.1
5-100	5	12.38%	-	5.6	5	8.31%	-	0.6	5	0.00%	-	0.2
8-1	5	41.32%	-	2.0	5	3.33%	-	0.1	5	0.00%	-	0.1
8-30	5	48.89%	-	140.9	5	17.68%	-	1.1	5	1.43%	-	0.1
8-100	3	47.23%	22.50%	113.0	5	21.08%	-	8.7	5	2.50%	-	0.4
10-1	5	45.08%	-	48.3	5	32.36%	-	3.5	5	2.16%	-	0.1
10-30	0	61.52%	20.64%	-	5	56.55%	-	39.5	5	8.45%	-	0.4
10-100	0	64.47%	50.91%	-	3	54.82%	7.50%	151.7	5	12.73%	-	1.5
12-1	1	67.13%	14.30%	33.2	5	40.60%	-	327.3	5	7.86%	-	0.5
12-30	0	88.61%	46.74%	-	5	42.93%	-	160.9	5	3.16%	-	0.8
12-100	0	84.37%	68.24%	-	5	51.54%	-	583.7	5	13.91%	-	1.7
15-1	2	60.11%	11.21%	369.6	3	53.01%	5.56%	410.0	5	11.57%	-	0.9
15-30	0	85.29%	65.29%	-	0	66.72%	22.66%	-	3	18.03%	4.74%	120.2
15-100	0	96.00%	86.92%	-	0	62.62%	24.58%	-	3	19.98%	6.45%	173.8

Table 2-5. Descriptions of the problem instances used for analyzing three-stage algorithm

Name	$\|N\|$	$\|K\|$	r	b
5	5	5	4	20
8	8	7	4	35
10	10	8	5	40
12	12	10	5	50
15	15	10	8	70
17	17	10	8	100
20	20	10	10	120
22	22	10	10	140

Our next experiment analyzes the performance of our three-stage algorithm for larger instances. For this experiment, we generated additional random problem instances using the parameter settings given in Table 2-5. Similar to our previous experiments, we

Table 2-6. Three-Stage algorithm on graphs having edge density = 0.2

	$\epsilon, \delta = 0.05$						$\epsilon, \delta = 0.01$					
Name	$\|Q\|$	Solved	Root Gap	Final Gap	Time	Heuristic Gap	$\|Q\|$	Solved	Root Gap	Final Gap	Time	Heuristic Gap
5	407	5	0.00%	-	0.8	2.86%	2194	5	0.00%	-	3.5	0.00%
8	837	5	0.00%	-	2.2	1.54%	4343	5	0.00%	-	12.7	3.33%
10	1169	5	10.88%	-	5.6	5.09%	6006	5	1.33%	-	19.6	1.33%
12	1724	5	1.11%	-	13.2	4.19%	8779	5	3.00%	-	58.5	3.33%
15	2140	5	7.24%	-	22.1	2.74%	10858	5	12.41%	-	170.9	4.37%
17	2417	5	10.19%	-	41.0	4.78%	12245	5	9.82%	-	211.1	8.01%
20	2833	5	16.55%	-	79.8	7.01%	14324	5	12.40%	-	403.7	4.51%
22	3110	5	14.77%	-	128.2	6.49%	15710	5	15.78%	-	699.9	6.46%

generated problem instances for which the expected edge density of the graph takes values 0.2, 0.3, and 0.4. For each data set, we calculated the number of scenarios corresponding to $\epsilon, \delta = 0.05$ and $\epsilon, \delta = 0.01$ using Proposition 1. Hence, inequality (2–14) ensures that we can be 95% (99%, respectively) certain that all demands can be satisfied 95% (99%, respectively) of the time. We generated five random instances for each data set,

Table 2-7. Three-Stage algorithm on graphs having edge density = 0.3

	$\epsilon, \delta = 0.05$						$\epsilon, \delta = 0.01$					
Name	$\|Q\|$	Solved	Root Gap	Final Gap	Time	Heuristic Gap	$\|Q\|$	Solved	Root Gap	Final Gap	Time	Heuristic Gap
5	407	5	0.00%	-	0.8	2.86%	2194	5	0.00%	-	3.5	0.00%
8	837	5	0.00%	-	2.6	2.86%	4343	5	3.33%	-	13.0	2.86%
10	1169	5	6.58%	-	7.5	8.99%	6006	5	11.86%	-	27.4	4.80%
12	1724	5	8.61%	-	16.1	4.51%	8779	5	8.22%	-	71.7	4.31%
15	2140	5	15.45%	-	45.1	3.05%	10858	4	18.53%	3.45%	176.4	4.25%
17	2417	5	13.63%	-	42.4	3.43%	12245	5	9.93%	-	189.3	2.68%
20	2833	5	17.47%	-	362.5	3.24%	14324	5	18.13%	-	639.1	3.32%
22	3110	4	16.18%	4.76%	159.3	5.82%	15710	5	15.86%	-	738.5	3.45%

resulting in 240 instances in total. In addition to the columns given in Table 2-2, Tables 2-6, 2-7, and 2-8 show the relative gap between the quality of the solution found by our initial heuristic (Section 2.3.5.2) and the best lower bound obtained ("Heuristic Gap"). Our algorithm can solve 206 instances out of 240 to optimality, and provides an average

Table 2-8. Three-Stage algorithm on graphs having edge density = 0.4

	$\epsilon, \delta = 0.05$						$\epsilon, \delta = 0.01$					
Name	$\|Q\|$	Solved	Root Gap	Final Gap	Time	Heuristic Gap	$\|Q\|$	Solved	Root Gap	Final Gap	Time	Heuristic Gap
5	407	5	0.00%	-	0.8	2.22%	2194	5	0.00%	-	3.2	0.00%
8	837	5	7.71%	-	2.7	2.43%	4343	5	7.25%	-	17.3	5.33%
10	1169	5	16.38%	-	9.4	5.71%	6006	5	15.84%	-	52.1	9.73%
12	1724	5	16.71%	-	63.2	5.41%	8779	5	14.13%	-	118.4	5.45%
15	2417	4	16.24%	2.86%	338.8	4.07%	12245	2	16.55%	4.71%	549.8	6.86%
17	2140	1	23.46%	8.46%	993.7	9.23%	10858	1	24.77%	11.44%	1515.5	13.07%
20	2833	0	18.83%	9.46%	-	9.46%	14324	0	20.01%	11.30%	-	11.81%
22	3110	0	18.36%	11.05%	-	11.47%	15710	0	17.83%	9.90%	-	10.29%

optimality gap of 9.21% for the 34 instances that it cannot solve to optimality. The maximum optimality gap obtained for the entire data set is 21.22%. The results also suggest that our heuristic for finding an initial feasible solution is quite effective: the average optimality gap for our heuristic is 4.97%, and the maximum optimality gap is 22.72%. Since these calculations are based on the lower bounds obtained for the problem instances that could not be solved to optimality, our reported gaps possibly overestimate the true gap between heuristic and optimal objective values.

CHAPTER 3
CONSECUTIVE-ONES MATRIX DECOMPOSITION PROBLEM

3.1 Introduction and Literature Survey

Cancer is one of the leading causes of death throughout the world. In the last century, external beam radiation therapy has emerged as a very important and powerful modality for treating many forms of cancer, either in primary form or in conjunction with other treatment modalities such as surgery, chemotherapy, or medication. In the United States today, approximately two-thirds of all newly diagnosed cancer patients receive radiation therapy for treatment. Since the radiation beams employed in radiation therapy damages all cells traversed by the beams, both in targeted areas in the patient that contain cancerous cells as well as any cells in healthy organs and tissues, the treatment must be carefully designed. This can partially be achieved by delivering radiation from several different directions, also called beam orientations. Therefore, patients receiving radiation therapy are typically treated on a clinical radiation-delivery device that can rotate around the patient. The most common device is called a linear accelerator and is typically equipped with a so-called *multileaf collimator (MLC)* system which can be used to judiciously shape the beams by forming apertures, thereby providing a high degree of control over the dose distribution that is received by a patient (see Figure 3-1 [1]). This

(a) 

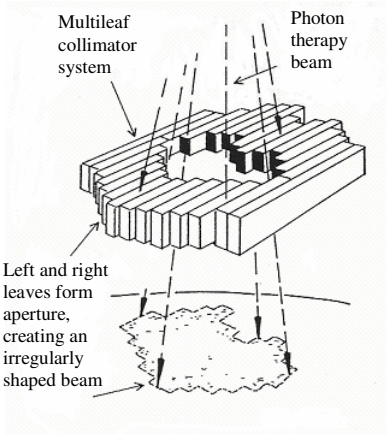
(b) 

Figure 3-1. (a) A multileaf collimator system (b) The projection of an aperture onto a patient

technique has been named *intensity modulated radiation therapy (IMRT)*.

Since the mid 1990's, large-scale optimization of the fluence applied from a number of beam orientations around a patient has been used to design treatments from MLC-equipped linear accelerators. A typical approach to IMRT treatment planning is to first select the number and orientations of the beams to use as well as an intensity profile or *fluence map* for each of these beams, where the fluence map takes the form of a matrix of intensities. This problem has been studied extensively and can be solved satisfactorily, in particular when (as is common in clinical practice) the beam orientations are selected manually by the physician or clinician based on their insight and expertise regarding treatment planning. For optimization approaches to the fluence map optimization problem with fixed beam orientations we refer to the review paper by Shepard et al. (1999). More recently, Romeijn et al. (2006) proposed new convex programming models, and Hamacher and Küfer (2002) and Küfer et al. (2003) considered a multi-criteria approach to the problem. Lee et al. (2000a, 2003) studied mixed-integer programming approaches to the extension of the fluence map optimization problem that also optimizes the number and orientations of the beams to be used. However, to enable delivery of the optimal fluence maps by the MLC system, they need to be decomposed into a collection of deliverable apertures. (For examples of integrated approaches to fluence map optimization, also referred to as aperture modulation, we refer to Shepard et al. (2002), Preciado-Walters et al. (2004), and Romeijn et al. (2005).)

The vast majority of MLC systems contain a collection of leaves that can be moved in parallel, thereby blocking part of the radiation beam. This architecture implies that we can view each beam as a matrix of beamlets or *bixels* (the smallest deliverable square beam that can be created by the MLC), so that each aperture can be represented by a collection of rows (or, by rotating the MLC head, columns) of bixels, each of which

[1] Varian Medical Systems; `http://www.varian.com/orad/prd056.html`.

should be convex. In other words, each fluence map should be decomposed into either constant-intensity row-convex apertures or constant-intensity column-convex apertures. Due to the time required for setup and verification, clinical practice prohibits using both types of apertures for a given fluence map, so that without loss of generality we focus on row-convex apertures only. Note that while some manufacturers of MLC systems impose additional constraints on the apertures, we assume that all row-convex apertures are deliverable. As an example, consider the fluence map given by the following 2×3 matrix of bixel intensities (see Baatar et al. (2005)):

$$\begin{bmatrix} 3 & 6 & 4 \\ 2 & 1 & 5 \end{bmatrix}.$$

If we represent an aperture by a binary matrix in which an element is equal to one if and only if the associated bixel is exposed (i.e., not blocked by either the left or right leaf of the MLC system), row-convexity corresponds to the property that, in each row of the corresponding matrix, the elements that are equal to one are consecutive (often referred to as the *consecutive-ones property*). Now observe that this fluence map can be decomposed into three apertures with corresponding intensities:

$$\begin{bmatrix} 3 & 6 & 4 \\ 2 & 1 & 5 \end{bmatrix} = 1 \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} + 2 \times \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} + 4 \times \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

Since, in general, there are many ways of decomposing a given fluence map into row-convex apertures, it is desirable to select the decomposition that can be delivered most efficiently. The two main efficiency criteria that play a role are the total *beam-on-time*, i.e., the total amount of time that the patient is being irradiated, and the *total setup time*, i.e., the total amount of time that is spent shaping the apertures. The former metric is proportional to the sum of intensities used in the decomposition, while the latter is approximately proportional to the number of matrices used in the decomposition. Although closely related, these two efficiency criteria are not equivalent. The example

given above shows the unique decomposition using only three apertures and with a beam-on-time of 7. However, the minimum beam-on-time for this fluence map is 6, which can be realized by four apertures using the following decomposition:

$$\begin{bmatrix} 3 & 6 & 4 \\ 2 & 1 & 5 \end{bmatrix} = 1 \times \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} + 1 \times \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} + 2 \times \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} + 2 \times \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

The problem of decomposing a fluence map while minimizing beam-on-time is polynomially solvable and has been widely studied, leading to several solution approaches for this problem. Bortfeld et al. (1994) proposed the sweep method, which Ahuja and Hamacher (2005) (who derived an equivalent method) showed to indeed yield an optimal solution; other exact algorithms were proposed by Kamath et al. (2003), and Siochi (1999). In addition, Baatar et al. (2005), Boland et al. (2004), Kalinowski (2005a), Kamath et al. (2004a,b,c,d), Lenzen (2000), and Siochi (1999) studied the problem of minimizing beam-on-time under additional hardware constraints, while Kalinowski (2005b) studied the benefits of allowing rotation of the MLC head.

Although the time required by the MLC system to transition between apertures formally depends on the apertures themselves, the fact that these times are similar and the presence of significant (aperture-independent) verification and recording overhead times justifies the use of the total number of setups (or, equivalently, the total number of apertures) to measure the total setup time. In addition, delivering IMRT with a small number of apertures provides the additional benefits of less wear-and-tear on the collimators (less stopping and starting) and a less error-prone delivery as IMRT delivery errors are known to be proportional to the number of apertures (see Stell et al. (2004)). The problem of decomposing a fluence map into the minimum number of row-convex apertures has been shown to be strongly NP-hard (see Baatar et al. (2005)), leading to the development of a large number of heuristics for solving this problem. Notable examples are the heuristics proposed by Baatar et al. (2005) (who also identify some polynomially

solvable special cases), Agazaryan and Solberg (2003), Dai and Zhu (2001), Que (1999), Que et al. (2004), Siochi (1999, 2004, 2007), Van Santvoort and Heijmen (1996), Xia and Verhey (1998). In addition, Engel (2005), Kalinowski (2005a), and Lim and Choi (2007) developed heuristics to minimize the number of apertures while constraining the total beam-on-time to be minimal. Finally, Langer et al. (2001) developed a mixed-integer programming formulation of the problem, while Kalinowski (2004) proposed an exact dynamic programming approach for the related problem of minimizing the number of apertures that yields the minimum beam-on-time. Baatar et al. (2007) described integer programming and constraint programming models for the same problem, and Ernst et al. (2009) proposed a constraint programming approach for minimizing the number of apertures. However, computational studies reported in Baatar et al. (2007); Ernst et al. (2009); Kalinowski (2004); Langer et al. (2001) reveal that these approaches can only be used to efficiently solve small problem instances to optimality. Our primary contribution is that we develop the first algorithm capable of solving clinical problem instances to optimality (or to provably near-optimality) within clinically acceptable computational time limits.

In this chapter, our focus is on the problem of finding a decomposition of a fluence map into row-convex apertures that minimizes total treatment time, as measured by the sum of the total setup time and beam-on-time. In Section 3.2 we develop our decomposition-based solution approach. In Section 3.3 we discuss the application of our algorithm on a collection of clinical and randomly generated test data, and compare its performance with alternative exact and heuristic techniques.

3.2 Decomposition Algorithm

Throughout this chapter, we denote the fluence map to be delivered by a matrix $B \in \mathbb{N}^{m \times n}$, where the element at row i and column j, (i, j), corresponds to a bixel with required intensity b_{ij}. Let w_1 be the time required by the MLC to form an aperture and w_2 denote the time required for the delivery of one unit of fluence. We refer to the

problem of minimizing the total treatment time, i.e., the sum of the aperture transition times and the total delivery time, as the *optimal leaf sequencing problem*.

We start this section by describing a decomposition framework for the optimal leaf sequencing problem in Section 3.2.1 and use this to formulate our master problem in Section 3.2.2. We introduce our subproblem in Section 3.2.3, prove its complexity, and provide a combinatorial search algorithm for its solution. We then enhance the empirical performance of our decomposition algorithm by introducing classes of valid inequalities to the master problem in Section 3.2.4, and finally describe an algorithm for constructing a feasible solution with medically desired properties in Section 3.2.5.

3.2.1 Decomposition Framework

To establish motivation for our approach, observe that if the objective is to minimize beam-on-time, the optimal leaf sequencing problem is decomposable by the rows of the fluence map. In particular, if the beam-on-time is minimized for each bixel row, the maximum of the corresponding beam-on-time values is equal to the minimum beam-on-time for the overall fluence map (see, e.g., Ehrgott et al. (2008)). However, this approach is not directly applicable when the objective is to minimize the total treatment time.

Even though the optimal leaf sequencing problem is not directly decomposable by rows, the fact that leaves corresponding to different rows can be positioned independently can still be exploited. Denote a particular positioning of left and right leaves for a row as a *leaf position*; an aperture is composed of a leaf position for each row of B. Our main observation is that *given* a collection of intensities, which can be used in apertures that collectively cover the fluence map, the rows are independent of one another. That is, we can determine the leaf positions to be used for covering each row independently, and then form apertures for covering the entire fluence map by combining individual leaf positions for each row that are assigned to the same intensity.

We define an *allowable intensity multiset* to be a collection of (potentially non-unique) intensity values, each of which can be assigned to a single aperture in our solution. We say that an allowable intensity multiset is *compatible* with a row if there exists a feasible decomposition of the row into leaf positions using a subset of that allowable intensity multiset. If an allowable intensity multiset is compatible with all rows, then it corresponds to a feasible decomposition of the fluence map and we call it a *feasible intensity multiset*. As an example, consider the fluence map given by the following 3×3 matrix:

$$B = \begin{bmatrix} 1 & 4 & 8 \\ 3 & 8 & 5 \\ 4 & 5 & 3 \end{bmatrix}. \tag{3–1}$$

Consider the allowable intensity multiset $\{1, 3, 5\}$. Assigning each of these values to at most one leaf position, the first row can be decomposed as

$$[1 \;\; 4 \;\; 8] \;\; = \;\; 1 \times [1 \;\; 1 \;\; 0] + 3 \times [0 \;\; 1 \;\; 1] + 5 \times [0 \;\; 0 \;\; 1]\,, \tag{3–2}$$

so that the allowable intensity multiset is compatible with the first row. Similarly, the second row can be decomposed as

$$[3 \;\; 8 \;\; 5] \;\; = \;\; 3 \times [1 \;\; 1 \;\; 0] + 5 \times [0 \;\; 1 \;\; 1]\,. \tag{3–3}$$

However, the first bixel in the third row must be covered by two leaf positions assigned to intensities 1 and 3, and the second bixel must be covered by a single leaf position assigned to intensity 5. Therefore, all allowable intensities must be used to cover the first two bixels, and the third bixel with required intensity 3 cannot be covered. Hence, the allowable intensity multiset is not compatible with the third row. Alternatively, consider an allowable intensity multiset that contains the values 1, 3, and 4 for the same fluence map. The rows can be decomposed as

$$[1 \;\; 4 \;\; 8] \;\; = \;\; 1 \times [1 \;\; 1 \;\; 1] + 3 \times [0 \;\; 1 \;\; 1] + 4 \times [0 \;\; 0 \;\; 1]\,,$$

$$\begin{aligned} [3 \;\; 8 \;\; 5] &= 1 \times [0 \;\; 1 \;\; 1] + 3 \times [1 \;\; 1 \;\; 0] + 4 \times [0 \;\; 1 \;\; 1]\,, \text{ and} \\ [4 \;\; 5 \;\; 3] &= 1 \times [0 \;\; 1 \;\; 0] + 3 \times [0 \;\; 0 \;\; 1] + 4 \times [1 \;\; 1 \;\; 0]\,. \end{aligned} \tag{3–4}$$

Since the allowable intensity multiset is compatible with all rows, it is a feasible intensity multiset having three leaf positions and a beam-on-time of 8. Furthermore, observe that the intensity requirements of the bixels in the first row strictly increase from left to right, implying that a leaf position must start at each bixel. Thus, any feasible decomposition of the first row uses at least three leaf positions, which yields a lower bound on the number of apertures. Also, the largest element of B is 8, which yields a lower bound on the beam-on-time. Since the given decomposition achieves the lower bounds on both objectives, we have an optimal solution to the optimal leaf sequencing problem.

3.2.2 Master Problem Formulation and Solution Approach

We represent an allowable intensity multiset by an integer vector $\mathbf{x} = (x_1, \ldots, x_L)$, where $L = \max_{i=1,\ldots,m;\, j=1,\ldots,n} b_{ij}$ is the maximum intensity value in the fluence map, and where x_ℓ is the number of times that intensity value ℓ occurs in the allowable intensity multiset. It is easy to see that, assuming all allowable intensity values are used, the number of apertures and the beam-on-time are, respectively, equal to

$$\sum_{\ell=1}^{L} x_\ell \text{ and } \sum_{\ell=1}^{L} \ell x_\ell. \tag{3–5}$$

The master problem can therefore succinctly be written as

$$\text{minimize } w_1 \sum_{\ell=1}^{L} x_\ell + w_2 \sum_{\ell=1}^{L} \ell x_\ell \tag{3–6}$$

subject to

$$\mathbf{x} \quad \text{is compatible with row } i \qquad \forall\, i = 1, \ldots, m \tag{3–7}$$

$$x_\ell \quad \text{integer} \qquad \forall\, \ell = 1, \ldots, L. \tag{3–8}$$

Clearly, our model contains the problem of minimizing the number of apertures as a special case by setting $w_1 = 1$ and $w_2 = 0$. Moreover, if we wish to minimize the number of apertures required while limiting the beam-on-time to no more than $\tilde{T}$, we simply add the following constraint to the model:

$$\sum_{\ell=1}^{L} \ell x_\ell \leq \tilde{T}, \tag{3–9}$$

where of course $\tilde{T}$ cannot be less than the minimum achievable beam-on-time $\tilde{z}$ (which can be found in polynomial time using the algorithms mentioned in Section 3.1).

To formulate our master problem as an integer programming problem, we introduce binary variables $y_{\ell r}$, $\forall\ \ell = 1, \ldots, L$, $r = 1, \ldots, R_\ell$, where $y_{\ell r} = 1$ if and only if $x_\ell = r$, and R_ℓ is an upper bound on the number of apertures having intensity ℓ used in an optimal solution. (We can compute R_ℓ by computing an initial upper bound on the optimal objective function value via any of the heuristics mentioned in Section 3.1, and then setting R_ℓ to the largest value such that $w_1 R_\ell + w_2 \ell R_\ell$ is no more than this bound.) Using these decision variables, we can reformulate the master problem (MP) as follows:

$$\text{minimize } w_1 \sum_{\ell=1}^{L} x_\ell + w_2 \sum_{\ell=1}^{L} \ell x_\ell \tag{3–10}$$

subject to

$$\sum_{r=1}^{R_\ell} r y_{\ell r} = x_\ell \qquad \forall\ \ell = 1, \ldots, L \tag{3–11}$$

$$\sum_{r=1}^{R_\ell} y_{\ell r} \leq 1 \qquad \forall\ \ell = 1, \ldots, L \tag{3–12}$$

$$\mathbf{x} \quad \text{is compatible with row } i \qquad \forall\ i = 1, \ldots, m \tag{3–13}$$

$$x_\ell \quad \text{integer} \qquad \forall\ \ell = 1, \ldots, L \tag{3–14}$$

$$y_{\ell r} \quad \text{binary} \qquad \forall\ \ell = 1, \ldots, L,\ r = 1, \ldots, R_\ell. \tag{3–15}$$

We next formulate (3–13) as a set of linear inequalities by deriving valid inequalities that cut off precisely those vectors $\mathbf{x}$ that violate (3–13). To this end, consider a particular

allowable intensity multiset represented by $\hat{\mathbf{x}}$ that is incompatible with at least one row. It is then clear that we should only consider vectors $\mathbf{x}$ that are different from $\hat{\mathbf{x}}$ in at least one component. We can achieve this by imposing the following constraint:

$$\sum_{\ell=1}^{L} \sum_{\substack{r=1 \\ r \neq \hat{x}_\ell}}^{R_\ell} y_{\ell r} \geq 1. \tag{3–16}$$

Since all integer solutions except for $\hat{\mathbf{x}}$ satisfy (3–16), it is indeed a valid inequality. Constraint (3–16) can be tightened by observing that if the solution $\hat{\mathbf{x}}$ is incompatible with row i, then any solution $\mathbf{x}$ such that $x_\ell \leq \hat{x}_\ell$, $\forall\ \ell = 1, \ldots, L$, is also incompatible with row i. Therefore, we require that $\mathbf{x}$ contain at least one component that is larger than its corresponding component in $\hat{\mathbf{x}}$, which yields the stronger valid inequality

$$\sum_{\ell=1}^{L} \sum_{r=\hat{x}_\ell+1}^{R_\ell} y_{\ell r} \geq 1. \tag{3–17}$$

Constraint (3–17) can, in turn, be tightened further by explicitly considering the rows for which $\mathbf{x}$ is incompatible. Let $L_i = \max_{j=1,\ldots,n} b_{ij}$ be the maximum intensity in the fluence map for row i. By the same argument as above, if the current solution $\hat{\mathbf{x}}$ is incompatible with row i, then any solution $\mathbf{x}$ such that $x_\ell \leq \hat{x}_\ell$, $\forall\ \ell = 1, \ldots, L_i$, is also incompatible with row i, since no leaf positions with intensity greater than L_i can be used in decomposing row i. Therefore, we require that $\mathbf{x}$ is larger than $\hat{\mathbf{x}}$ in at least one component $1, \ldots, L_i$:

$$\sum_{\ell=1}^{L_i} \sum_{r=\hat{x}_\ell+1}^{R_\ell} y_{\ell r} \geq 1 \quad \forall \text{ rows } i \text{ incompatible with } \hat{\mathbf{x}}. \tag{3–18}$$

Since (3–18) is stronger than (3–16) or (3–17), we use the latter inequalities in our model. Note also that (3–18) stated for row i_1 dominates a cut generated for row i_2 if $L_{i_1} < L_{i_2}$. Thus, we consider the bixel rows in nondecreasing order of their L_i-values, halt when an infeasible row is detected, and add a single inequality of the form (3–18). This sequence also tends to minimize subproblem execution time, since rows having a small maximum

intensity are easier to solve by the nature of the backtracking algorithm discussed in Section 3.2.3.

Since the collection (3–18) contains an exponential number of valid inequalities, we add them only as needed in a cutting plane fashion. In particular, this means that we relax (3–18), solve the relaxation of (MP) and generate an $\mathbf{x}$-solution representing a candidate allowable intensity multiset. We then solve a subproblem for each bixel row to determine if the allowable intensity multiset is incompatible with that row. If not, we have found an optimal solution to (MP). Otherwise, we add a constraint of the form (3–18) to (MP) that cuts off that solution.

3.2.3 Subproblem Analysis and Solution Approach

In this section, we consider the subproblem of checking whether a given intensity multiset $\mathbf{x}$ is compatible with a particular bixel row. For convenience and wherever the interpretation is clear from the context, we suppress the index i of the bixel row and denote a typical row of the fluence map B by $\mathbf{b} = (b_1, \ldots, b_n)$.

We represent a feasible decomposition as a collection of n-dimensional binary vectors $\mathbf{v}_{\ell r}$ ($\ell = 1, \ldots, L$; $r = 1, \ldots, x_\ell$). The values of $\mathbf{v}_{\ell r}$ that equal 1 correspond to the (consecutive) exposed bixels in the r^{th} aperture having intensity ℓ. For example, the decomposition in equation (3–2) corresponds to $\mathbf{v}_{11} = (1, 1, 0)$, $\mathbf{v}_{31} = (0, 1, 1)$, $\mathbf{v}_{51} = (0, 0, 1)$, and $\mathbf{v}_{\ell r} = \mathbf{0}$ for other ℓ, r. (Note that this decomposition would be feasible as long as $x_1, x_3, x_5 \geq 1$.) The subproblem can then formally be presented as follows:

C1-Partition

INSTANCE: An n-dimensional vector of nonnegative integers $\mathbf{b}$ and an integer vector $\mathbf{x} = (x_1, \ldots, x_L)$.

QUESTION: Do there exist n-dimensional binary vectors $\mathbf{v}_{\ell r}$ ($\ell = 1, \ldots, L$; $r = 1, \ldots, x_\ell$) that satisfy the consecutive-ones property such that $\sum_{\ell=1}^{L} \sum_{r=1}^{x_\ell} \ell \mathbf{v}_{\ell r} = \mathbf{b}$?

Proposition 3. C1-Partition *is strongly NP-complete.*

Proof. See Appendix B. □

In principle, the C1-PARTITION problem can be formulated and solved as an integer program. However, we have developed a computationally more effective backtracking algorithm that focuses on partitioning intensity requirements individually for each bixel. An integer vector $\mathbf{p}^j = (p_1^j, \ldots, p_L^j)$ provides a *bixel decomposition* of bixel $j \in \{1, \ldots, n\}$ in row $\mathbf{b}$ if and only if $b_j = \sum_{\ell=1}^{L} \ell p_\ell^j$. We then attempt to form a collection of leaf positions that realizes the individual bixel partitions. We call such a collection of leaf positions a *leaf decomposition* of $\mathbf{b}$.

To more effectively conduct our subproblem searches, we describe a property that holds in some leaf decomposition (if one exists) that satisfies the given collection of bixel decompositions.

Lemma 1. *Consider candidate bixel decompositions for bixels j and $j+1$, for some $j \in \{1, \ldots, n-1\}$, and suppose that these have a common decomposition intensity value ℓ, i.e., $p_\ell^j, p_\ell^{j+1} > 0$. Then, if a leaf decomposition exists, one exists in which a leaf position having intensity ℓ exposes both bixels j and $j+1$.*

Proof. Assume that there exists a leaf decomposition $\mathcal{V}$ in which bixels j and $j+1$ are exposed by two separate leaf positions, $\mathbf{v}_1$ and $\mathbf{v}_2$, respectively, each having intensity ℓ. Now consider the leaf position $\mathbf{v}_3 = \mathbf{v}_1 + \mathbf{v}_2$ having intensity ℓ. Then $\mathcal{V}' = \{\mathbf{v}_3\} \cup \mathcal{V} \setminus \{\mathbf{v}_1, \mathbf{v}_2\}$ is also a leaf decomposition that realizes the given bixel decomposition. □

We next derive a necessary condition that any feasible bixel decomposition must satisfy so that the corresponding set of leaf positions is compatible with a given allowable intensity multiset $\mathbf{x}$. Similar to the idea behind Lemma 1, if $p_\ell^j > p_\ell^{j+1}$, then $p_\ell^j - p_\ell^{j+1}$ leaf positions having intensity ℓ must expose bixel j but not $j+1$. Lemma 2 formalizes this idea.

Lemma 2. *Let $\mathbf{x}$ represent an allowable intensity multiset, and $\mathbf{p}^{j_\eta}$ denote candidate bixel decompositions for bixels j_η, $\forall \eta = 1, \ldots, n'$ such that $1 \leq j_1 < \cdots < j_{n'} \leq n$. The following*

set of conditions must be satisfied in any feasible solution.

$$\sum_{\eta=2}^{n'} \max\{0, p_\ell^{j_{\eta-1}} - p_\ell^{j_\eta}\} + p_\ell^{j_{n'}} \leq x_\ell \quad \forall \ell = 1, \ldots, L. \tag{3–19}$$

Proof. If $p_\ell^{j_{\eta-1}} > p_\ell^{j_\eta}$, at least $p_\ell^{j_{\eta-1}} - p_\ell^{j_\eta}$ leaf positions having intensity ℓ must expose bixel $j_{\eta-1}$ but not j_η. Also, at least $p_\ell^{j_{n'}}$ leaf positions having intensity ℓ must expose bixel $j_{n'}$. Since all leaf positions listed above are necessarily disjoint, the lemma holds. □

We next describe our backtracking algorithm. In this algorithm, we first enumerate all possible ways of decomposing the bixel intensities in $\mathbf{b}$ using a subset of the allowable intensity multiset given by $\mathbf{x}$. We denote the set of all candidate bixel decompositions for bixel j by $\mathcal{P}_j$, where for each $\mathbf{p} \in \cup_{j=1}^{n} \mathcal{P}_j$, we must have $p_\ell \leq x_\ell, \ \forall \ \ell = 1, \ldots, L$.

The backtracking algorithm for solving the subproblem is stated formally in Algorithm 1. We begin by enumerating each possible element of $\mathcal{P}_j, \forall\ j\ =\ 1, \ldots, n$. We denote the set of processed bixels by $\mathcal{F}$ (for which a candidate "active" bixel decomposition has been established), and the set of unprocessed bixels by $\mathcal{R}$. In each iteration, we check to see if the set of candidate bixel decompositions $\mathcal{P}_j$ for any $j \in \mathcal{R}$ is empty. If so, the current active bixel decompositions do not yield a feasible solution, and the algorithm backtracks. Otherwise, we consider an unprocessed bixel $\hat{\jmath} \in \mathcal{R}$, and choose an untried bixel decomposition $\mathbf{p}^{\hat{\jmath}} \in \mathcal{P}_{\hat{\jmath}}$ to be active for bixel $\hat{\jmath}$. Next, we move $\hat{\jmath}$ from $\mathcal{R}$ to $\mathcal{F}$, creating updated sets $\mathcal{R}'$ and $\mathcal{F}'$, and invoke Lemma 2 to update the set of bixel decompositions for the bixels in $\mathcal{R}'$. Specifically, for each $j \in \mathcal{R}'$ and $\mathbf{p}^j \in \mathcal{P}_j$, we calculate the number of leaf positions that would be required due to selecting $\mathbf{p}^j$ as the active bixel decomposition for bixel j, in addition to those already selected for bixels in $\mathcal{F}'$. We eliminate $\mathbf{p}^j$ if a condition of type (3–19) is violated. We then recursively call the procedure to continue with a new bixel $j' \in \mathcal{R}'$.

We stop either when we find a feasible bixel decomposition for all bixels, or when we exhaust all bixel decompositions without finding a feasible solution. In the former case, a leaf decomposition that realizes the bixel decompositions for bixels $j \in \{1, \ldots, n\}$ can be

found by invoking Algorithm 3, which is based on the repeated application of Lemma 1. To see that Algorithm 3 recovers a feasible leaf decomposition, note that Algorithms 1 and 2 provide bixel decompositions that satisfy Lemma 2, and in particular, the condition

$$\sum_{j=2}^{n} \max\{0, p_\ell^{j-1} - p_\ell^j\} + p_\ell^n \leq x_\ell \quad \forall \ell = 1, \ldots, L. \tag{3–20}$$

Algorithm 3 recovers a feasible leaf decomposition if, in the outer while-loop corresponding to each $\ell = 1, \ldots, L$, the counter r is never incremented more than x_ℓ times. Note that r is incremented each time the inner while-loop terminates, which occurs either when $\tilde{j} > n$ (a total of p_ℓ^n times), or when $p_\ell^{\tilde{j}} = 0$ ($p_\ell^{\tilde{j}-1} - p_\ell^{\tilde{j}}$ times) for $\tilde{j} = 2, \ldots, n$. The total number of times that r is incremented in the outer while-loop for $\ell = 1, \ldots, L$ is thus the left-hand-side of (3–20), which is no more than x_ℓ, as required.

If we exhaust all bixel decompositions without finding a feasible solution, we conclude that the current allowable intensity multiset is incompatible with the current row.

Algorithm 1 C1-PARTITION($\mathbf{b}$, $\mathbf{x}$)

Input: $\mathbf{b}$ {n-dimensional vector representing bixel intensity requirements}
Input: $\mathbf{x}$ {L-dimensional vector representing an allowable intensity multiset}
{This algorithm finds whether there exists a C1-PARTITION of $\mathbf{b}$ compatible with $\mathbf{x}$}
$\mathcal{F} \leftarrow \emptyset$ {$\mathcal{F}$ is the set of processed bixels}
$\mathcal{R} \leftarrow \{1, \ldots, n\}$ {$\mathcal{R}$ is the set of unprocessed bixels}
for all $j \in \{1, \ldots, n\}$ **do**
  $\mathcal{P}_j \leftarrow$ Enumerate all bixel decompositions compatible with $\mathbf{x}$ for bixel j
$\mathcal{P} \leftarrow \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$
return C1-PARTITIONRECURSIVE($\mathbf{b}$, $\mathbf{x}$, $\mathcal{F}$, $\mathcal{R}$, $\mathcal{P}$)

Since Algorithm 1 is a backtracking algorithm, and therefore in the worst case investigates all possible bixel decompositions, it is of exponential time complexity (as expected, due to Proposition 3). However, the empirical running time of the algorithm can be reduced using the following observations:

(i) If two adjacent bixels in a row have the same required intensity value, there must exist an optimal solution in which they are exposed by the same leaf positions. This result can be proven in a similar way as Lemma 1, and is therefore omitted for brevity. This observation implies that we can preprocess the data by merging

Algorithm 2 C1-PartitionRecursive($\mathbf{b}$, $\mathbf{x}$, $\mathcal{F}$, $\mathcal{R}$, $\mathcal{P}$)

if $\mathcal{R} = \emptyset$ **then**
  return true {all bixels have been processed, $\mathcal{P}$ represents a feasible solution}
else
  if $\exists j \in \mathcal{R} : \mathcal{P}_j = \emptyset$ **then**
    return false {there is no remaining way of decomposing bixel j}
else
  $\hat{\jmath} \leftarrow \text{argmin}_{j \in \mathcal{R}} |\mathcal{P}_j|$ {$\hat{\jmath}$ is a bixel having the smallest number of bixel decompositions}

  for all $\mathbf{p}^{\hat{\jmath}} \in \mathcal{P}_{\hat{\jmath}}$ **do**
    $\mathcal{P}' \leftarrow \mathcal{P}$, $\mathcal{P}'_{\hat{\jmath}} \leftarrow \{\mathbf{p}^{\hat{\jmath}}\}$ {$\mathbf{p}^{\hat{\jmath}}$ is now the active decomposition for bixel $\hat{\jmath}$}
    $\mathcal{F}' \leftarrow \mathcal{F} \cup \{\hat{\jmath}\}$, $\mathcal{R}' \leftarrow \mathcal{R} \setminus \{\hat{\jmath}\}$
    for all $j \in \mathcal{R}'$ **do**
      $\mathcal{P}'_j \leftarrow \mathcal{P}_j \setminus$ {all elements eliminated by Lemma 2, given the active decompositions $\mathbf{p}^{\tilde{\jmath}}$ for $\tilde{\jmath} \in \mathcal{F}'$}
    if C1-PartitionRecursive($\mathbf{b}$, $\mathbf{x}$, $\mathcal{F}'$, $\mathcal{R}'$, $\mathcal{P}'$) **then**
      return true {a feasible solution that uses $\mathbf{p}^{\hat{\jmath}}$ to decompose bixel $\hat{\jmath}$ is found}
  return false {all bixel decompositions of bixel $\hat{\jmath}$ have been exhausted}

Algorithm 3 RecoverLeafDecomposition($\mathbf{b}$, $\mathbf{x}$, $\mathcal{P}$)

Require: $\mathcal{P}_j = \{\mathbf{p}^j\}$ $\forall j \in \{1, \ldots, n\}$ {all bixels have been processed}
Output: $\mathbf{v}_{\ell r}$ $(\ell = 1, \ldots, L;\ r = 1, \ldots, x_\ell)$
{$\mathbf{v}_{\ell r}$ is an n-dimensional binary vector that represents a leaf position}
for all $\ell \in \{1, \ldots, L\}$, $r \in \{1, \ldots, x_\ell\}$ **do**
  $\mathbf{v}_{\ell r} \leftarrow \mathbf{0}$
for all $\ell \in \{1, \ldots, L\}$ **do**
  $r \leftarrow 1$, $j \leftarrow 1$
  while $j \leq n$ **do**
    if $p_\ell^j > 0$ **then**
      $\tilde{\jmath} \leftarrow j$ {a new leaf position must start at bixel j}
      while $\tilde{\jmath} \leq n$ and $p_\ell^{\tilde{\jmath}} > 0$ **do**
        {expand the new leaf position as much as possible}
        $v_{\ell r \tilde{\jmath}} \leftarrow 1$
        $p_\ell^{\tilde{\jmath}} \leftarrow p_\ell^{\tilde{\jmath}} - 1$, $\tilde{\jmath} \leftarrow \tilde{\jmath} + 1$
      $r \leftarrow r + 1$
    else
      $j \leftarrow j + 1$ {all leaf positions that start at bixel j have been recovered}

all adjacent bixels in a bixel row having the same intensity requirement, thereby reducing the dimensionality of the problem instance.

(ii) In choosing the next bixel to be processed, we pick a bixel $j \in \mathcal{R}$ having the smallest number of remaining candidate bixel decompositions. In this manner, we can quickly enumerate all possible bixel decompositions for a few key bixels and eliminate a significant portion of bixel decompositions for the remaining bixels without wasting effort by unnecessary backtracking steps.

(iii) In choosing the next candidate bixel decomposition $\mathbf{p}^j \in \mathcal{P}_j$ for a chosen bixel $j \in \mathcal{R}$, we select an untried bixel decomposition having the fewest number of intensity values. Since each intensity value used in decomposing a bixel needs to be assigned to a different aperture, this rule favors a bixel decomposition using the fewest number of apertures to decompose the chosen bixel. Therefore, it tends to retain the availability of more elements of the allowable intensity multiset (and hence apertures) for the remaining bixels, making it easier to find a feasible solution (if one exists).

3.2.4 Valid Inequalities for the Master Problem

The initial optimal solution to the relaxation of (MP) in which none of the inequalities (3–18) have yet been added to the model will set all variables equal to zero, which is clearly incompatible with all rows. In this section, we derive some characteristics of all feasible solutions and use these to define valid inequalities for (MP). In this way, we attempt to improve the convergence rate of the decomposition algorithm by eliminating some clearly infeasible solutions before the initial execution of the master problem.

3.2.4.1 Beam-on-time and number of apertures inequalities

Our first observation uses and generalizes the fact that the beam-on-time, number of apertures, and total treatment time required for the decomposition of any single row into leaf positions provide lower bounds on the minimum beam-on-time, number of apertures, and total treatment time, respectively, needed to deliver the entire fluence map. More generally, consider any collection of nonnegative objective weights w_1' and w_2' in place of w_1 and w_2, and let $T_i(w_1', w_2')$ be the minimum value of the objective with respect to these weights over all decompositions for row i only. Then the following are valid inequalities for

(MP):

$$w_1' \sum_{\ell=1}^{L} x_\ell + w_2' \sum_{\ell=1}^{L} \ell x_\ell \geq T_i(w_1', w_2') \quad \forall\, i = 1, \ldots, m. \tag{3–21}$$

We formulate an integer programming model to determine $T_i(w_1', w_2')$ for a given row i. First, denote the set of possible leaf positions for that row by $\mathcal{K}$, and define n-dimensional binary vectors $\mathbf{v}_k$ for $k \in \mathcal{K}$ (where $|\mathcal{K}| = O(n^2)$), such that $v_{kj} = 1$ if and only if bixel j is exposed by leaf position k. In addition to decision variables x_ℓ as in (MP), define binary decision variables $z_{k\ell}$, $\forall\, k \in \mathcal{K}$, $\ell = 1, \ldots, L^k$ such that $z_{k\ell} = 1$ if and only if leaf position k is used with intensity ℓ (where $L^k = \min_{j:v_{kj}=1} b_j$ is an upper bound on the intensity of leaf position k.) Then $T_i(w_1', w_2')$ is the optimal objective function value of the following optimization problem, (SR):

$$\text{minimize } w_1' \sum_{\ell=1}^{L} x_\ell + w_2' \sum_{\ell=1}^{L} \ell x_\ell \tag{3–22}$$

subject to

$$\sum_{k \in \mathcal{K}} \left(v_{jk} \sum_{\ell=1}^{L^k} \ell z_{k\ell} \right) = b_j \qquad \forall\, j = 1, \ldots, n \tag{3–23}$$

$$\sum_{\ell=1}^{L^k} z_{k\ell} \leq 1 \qquad \forall\, k \in \mathcal{K} \tag{3–24}$$

$$\sum_{k \in \mathcal{K}: L^k \geq \ell} z_{k\ell} = x_\ell \qquad \forall\, \ell = 1, \ldots, L \tag{3–25}$$

$$z_{k\ell} \in \{0, 1\} \qquad \forall\, k \in \mathcal{K},\ \ell = 1, \ldots, L^k \tag{3–26}$$

$$x_\ell \geq 0 \text{ and integer} \qquad \forall\, \ell = 1, \ldots, L. \tag{3–27}$$

Constraints (3–23) ensure that each bixel receives exactly its required amount of dose while constraints (3–24) guarantee that each leaf position is either not used or is assigned to a single intensity value. Finally, constraints (3–25) relate the x- and z-variables.

A practical difficulty in implementing the valid inequalities of the form (3–21) is that we must determine appropriate values for the weights w_1' and w_2'. However, Baatar (2005) shows that, when decomposing a single bixel row, there exists a set of leaf positions

that simultaneously minimizes both beam-on-time and the number of apertures. If we let $N_i = T_i(1,0)$ represent the minimum number of apertures for row i, and $\tilde{z}_i = T_i(0,1)$ represent the minimum beam-on-time for row i, this implies that $T_i(w_1', w_2') = w_1' N_i + w_2' \tilde{z}_i$, so that we can replace (3–21) by

$$w_1' \sum_{\ell=1}^{L} x_\ell + w_2' \sum_{\ell=1}^{L} \ell x_\ell \geq w_1' N_i + w_2' \tilde{z}_i \quad \forall\ i = 1, \ldots, m. \tag{3–28}$$

It is easy to see that we can capture *all* of these valid inequalities by restricting ourselves to the coefficient pairs $(w_1', w_2') = (1,0)$ and $(0,1)$ only:

$$\sum_{\ell=1}^{L} x_\ell \geq \max_{i=1,\ldots,m} \{N_i\} \tag{3–29}$$

$$\sum_{\ell=1}^{L} \ell x_\ell \geq \max_{i=1,\ldots,m} \{\tilde{z}_i\}. \tag{3–30}$$

We can generalize this idea as follows. Let $R(\mathcal{L})$ denote the set of rows for which the maximum intensity requirement is bounded by $\mathcal{L}$ for some $\mathcal{L} \in \{1, \ldots, L\}$, i.e., $R(\mathcal{L}) = \{i \in \{1, \ldots, m\} : L_i \leq \mathcal{L}\}$. Since intensity values greater than $\mathcal{L}$ cannot be used in decomposing the rows in $R(\mathcal{L})$, a similar approach to the one above can be used to derive the following family of valid inequalities

$$\sum_{\ell=1}^{\mathcal{L}} x_\ell \geq \max_{i \in R(\mathcal{L})} \{N_i\} \quad \forall\ \mathcal{L} = 1, \ldots, L \tag{3–31}$$

$$\sum_{\ell=1}^{\mathcal{L}} \ell x_\ell \geq \max_{i \in R(\mathcal{L})} \{\tilde{z}_i\} \quad \forall\ \mathcal{L} = 1, \ldots, L. \tag{3–32}$$

Finally, note that the values of N_i and $\tilde{z}_i$ can be found by solving (SR) with $w_1' = 1, w_2' = 1$ or by using the method of Kalinowski (2004), since there exists a solution that minimizes both beam-on-time and the number of apertures (Baatar, 2005).

3.2.4.2 Bixel subsequence inequalities

Recall that (3–16)–(3–18) represent necessary conditions for feasibility of an allowable intensity multiset with respect to a particular row. It is possible to develop stronger

necessary conditions if we examine subsequences of a row, i.e., a subset of the required intensity values in a row that preserves their order in the fluence map. First, Lemma 3 shows that, if a given allowable intensity multiset is incompatible with a subsequence $\mathbf{s}$ of row i, then it also must be incompatible with row i.

Lemma 3. *Consider an allowable intensity multiset* $\mathbf{x}$*, an* n*-dimensional vector* $\mathbf{b}$ *that represents the intensity requirements of the bixels in some row of* B*, and an* n'*-dimensional vector* $\mathbf{s} = (b_{j_1}, \ldots, b_{j_{n'}})$*, where* $1 \leq j_1 < \cdots < j_{n'} \leq n$*. If* $\mathbf{x}$ *is not compatible with* $\mathbf{s}$*, then it is also not compatible with* $\mathbf{b}$*.*

Proof. We prove the equivalent statement that if $\mathbf{x}$ is compatible with $\mathbf{b}$, then it is also compatible with $\mathbf{s}$. Assume that $\mathbf{x}$ is compatible with $\mathbf{b}$. By definition, there exists a bixel decomposition for each bixel $j = 1, \ldots, n$ so that the resulting set of leaf positions is compatible with $\mathbf{x}$. The bixel decompositions corresponding to only the bixels in $\mathbf{s}$ are also compatible with $\mathbf{x}$, since the order of the bixels in $\mathbf{s}$ is the same as that in $\mathbf{b}$. □

Note that we can invoke Lemma 3 to associate a subproblem with each of the $O(2^n)$ subsequences of a bixel row $\mathbf{b}$. Each of these subproblems can then be used to generate cutting planes of the form (3–18), as well as valid inequalities of the form (3–31) and (3–32). However, since the strength of (3–18), (3–31) and (3–32) depend on the largest intensity value in a bixel row, we form subsequences of each bixel row by, for $\mathcal{L} = 1, \ldots, L$, considering only those bixels having required intensity less than or equal to $\mathcal{L}$. The valid inequalities generated by the $O(\min(n, L))$ subsequences generated in this fashion imply *all* $O(2^n)$ valid inequalities associated with all possible subsequences.

3.2.5 Constructing a Feasible Matrix Decomposition

Our algorithm finds an optimal allowable intensity multiset and a bixel decomposition for each bixel row. To construct a corresponding matrix decomposition, we need to apply Algorithm 3 to find a leaf decomposition for each row. We can then generate aperture matrices by arbitrarily combining leaf positions using the same intensity values in different

rows. We have found empirically that this simple approach yields a feasible matrix decomposition very quickly.

Since any pair of leaf positions assigned to the same intensity value in different rows can be combined, there are up to $\left(\prod_{\ell=1}^{L}(x_\ell!)\right)^m$ aperture matrices that can be constructed from a given feasible leaf decomposition for each row. Even though each such choice represents an alternative optimal solution to the optimal leaf sequencing problem, some matrix decompositions may clinically be preferable to others based on their structural properties. Perhaps the most challenging structural consideration pertains to the so-called "tongue-and-groove" effect observed in MLCs. We refer the reader to the works of Deng et al. (2001) and Que et al. (2004) for technical details of the tongue-and-groove effect in dynamic MLC dose delivery. For the purposes of this study, it is sufficient to understand that leaves in adjacent rows often interlock with a tongue on the bottom of one row sliding along a groove in the top of another row. Tongue-and-groove underdosage occurs since a leaf's tongue blocks dosage intended for cells beneath it. Therefore, it is desirable to limit such underdosages.

To measure the amount of tongue-and-groove effect in a treatment plan, Que et al. (2004) note that it is generally not desirable to deliver one aperture in which some bixel (i, j) is blocked by a leaf while bixel $(i + 1, j)$ is not blocked, if another aperture is being delivered where (i, j) is not blocked by a leaf while $(i + 1, j)$ is blocked. Based on this observation, Que et al. (2004) derive the following *tongue-and-groove index* (TGI). Suppose a treatment plan consists of K apertures described by binary values v_j^{ik}, where $v_j^{ik} = 0$ if cell (i, j) is blocked by a leaf in aperture k and $v_j^{ik} = 1$ otherwise, for each $i = 1, \ldots, m$, $j = 1, \ldots, n$, $k = 1, \ldots, K$. Let I_k be the intensity delivered in aperture $k = 1, \ldots, K$. Then the TGI of a matrix decomposition is defined as:

$$\sum_{i=1}^{m-1}\sum_{j=1}^{n}\sum_{k=1}^{K-1}\sum_{\ell=k+1}^{K}\min\{I_k, I_\ell\}\left[v_j^{ik}(1-v_j^{i+1,k})(1-v_j^{i\ell})v_j^{i+1,\ell} + (1-v_j^{ik})v_j^{i+1,k}v_j^{i\ell}(1-v_j^{i+1,\ell})\right]. \quad (3\text{–}33)$$

We thus can calculate the TGI component induced by rows 1 and 2 (of all aperture pairs), then rows 2 and 3, and so on, down to rows $m-1$ and m. This observation allows us to focus on pairs of rows instead of pairs of entire aperture matrices while reducing TGI, allowing us to design an efficient algorithm for TGI reduction given a set of bixel decompositions for each row.

Given a pair of adjacent rows, we attempt to match individual leaf positions in the two rows to minimize the TGI induced by the adjacent row pair. To limit computational overhead in this phase of our algorithm, we reduce TGI indirectly by the following scheme. Let us denote a leaf position for row i by a binary n-vector $\mathbf{v}^i$, where $v^i_j = 1$ if the leaf position exposes bixel j in row i. We measure the *overlap* between two leaf positions having the same intensity value in consecutive rows by counting the number of columns that both leaf positions expose simultaneously. Formally, we define the overlap between leaf positions $\mathbf{v}^i$ and $\mathbf{v}^{i+1}$ as $\theta(\mathbf{v}^i, \mathbf{v}^{i+1}) = \sum_{j=1}^{n} v^i_j v^{i+1}_j$. Our approach is to heuristically minimize TGI by maximizing the total overlap between all leaf position pairs, which can efficiently be solved as an assignment problem. The efficiency of the assignment problems can be further improved by noting that the problem decomposes over the intensity values $\ell \in \{1, \ldots, L\}$, since only leaf positions having the same intensity value can be combined. Therefore, we can generate a matrix decomposition by finding a leaf decomposition for each row, and then matching leaf positions in adjacent rows having the same intensity value by solving an assignment problem so that the total overlap is maximized.

The TGI minimization step described in the previous paragraph can be improved as follows. Typically, multiple bixel decompositions exist for each row that are compatible with a given feasible intensity multiset. Algorithm 2 can be modified in a straightforward manner so that it finds *all* leaf decompositions of a row, instead of stopping once the first feasible bixel decomposition for all bixels is found. Since different bixel decompositions for a bixel row correspond to different leaf decompositions, considering alternative bixel decompositions can lead to a matrix decomposition having a smaller TGI.

Given alternative leaf decompositions for each row, the problem of minimizing TGI can be formulated as a shortest path problem as follows. We create a layered network in which each layer corresponds to a bixel row $i \in \{1, \ldots, m\}$, and node N_{id} represents the d^{th} leaf decomposition of row i. We add a directed arc from each node N_{id} to all nodes $N_{(i+1)d'}$, for all $i = 1, \ldots, m-1$. The cost of the arc from node N_{id} to $N_{(i+1)d'}$ is given by the TGI value resulting from the assignment solution corresponding to the candidate leaf decompositions represented by d for row i, and d' for row $i+1$. Finally, we add a start node S and a finish node F. We create zero-cost arcs from S to all nodes in the first layer, and from all nodes in the last layer to F. A shortest S–F path in this graph represents a matrix decomposition having a minimum TGI from among the provided options. Since the graph is acyclic, the shortest path problem can be solved in $O(|\mathcal{A}|)$ time, where $\mathcal{A}$ is the set of all arcs.

Remark 5. *The shortest path approach to minimizing TGI can be difficult to solve quickly when bixel rows have a large number of alternative leaf decompositions, since an arc joins each pair of nodes corresponding to adjacent bixel rows. To partially overcome this difficulty, we limit the number of bixel decompositions found by Algorithm 2 by terminating once 250 feasible bixel decompositions have been identified. Next, note that a straightforward acyclic shortest path implementation processes layers one at a time, and does not generate a feasible S–F path before processing the last layer. Since being able to specify a time limit is a desired feature in a practical setting, we use a hybrid algorithm for solving the shortest path problem. Our algorithm starts by processing layers one-by-one, updating node labels as usual. If a shortest path is not found when a given initial time limit expires, our algorithm switches to a depth-first-search (DFS) procedure, which we terminate after a given final time limit. We start DFS from an unprocessed node N_{id} having a smallest label, select a minimum-cost arc $(N_{id}, N_{(i+1)d'})$ exiting that node, and update the label of $N_{(i+1)d'}$ if we have found a new shortest S–$N_{(i+1)d'}$ path. Else, the*

algorithm backtracks and seeks another arc from N_{id}. We then return the shortest S–F path found by this procedure when the final time limit is reached.

3.3 Computational Results and Comparisons

3.3.1 Problem Instances

In our experiments we have used two classes of problem instances. Our base set of test problem instances consists of 25 clinical problem instances that were obtained from treatment plans for five head-and-neck cancer patients treated using five beam angles each. Table 3-1 reports the problem characteristics for these problem instances in terms of the matrix dimensions m and n. The maximum intensity value is $L = 20$ for all these instances. In addition, to allow comparison of our results with published results on other approaches to the problem, we generated 100 random problem instances of dimensions 20×20 having maximum intensity value $L = 10$.

However, since these problem instances are generally too large to be solvable by the integer programming model from Langer et al. (2001) and its modification described in Appendix A, we also randomly generated eight instances ("test5x5a", ..., "test6x7b") to demonstrate the computational limitations of the latter approaches. Unless otherwise specified, we used $w_1 = 7$ and $w_2 = 1$ as the objective weights for the number of apertures and beam-on-time, respectively.

Table 3-1. Dimensions of clinical problem instances

Name	m	n	Name	m	n	Name	m	n	Name	m	n	Name	m	n
c1b1	15	14	c2b1	18	20	c3b1	22	17	c4b1	19	22	c5b1	15	16
c1b2	11	15	c2b2	17	19	c3b2	15	19	c4b2	13	24	c5b2	13	17
c1b3	15	15	c2b3	18	18	c3b3	20	17	c4b3	18	23	c5b3	14	16
c1b4	15	15	c2b4	18	18	c3b4	19	17	c4b4	17	23	c5b4	14	16
c1b5	11	15	c2b5	17	18	c3b5	15	19	c4b5	12	24	c5b5	12	17

3.3.2 Implementation Issues

We have implemented our decomposition algorithm using CPLEX 11.0 running on a Windows XP PC with a 3.4 GHz CPU and 2 GB RAM. We use callback functions of CPLEX to generate a single branch-and-bound tree in which we solve the subproblems corresponding to each integer solution found in the tree, and add cuts to tighten the

master problem as necessary. This implementation turned out to be consistently faster than one which re-solves the master problem each time a cutting plane is added to the model. Furthermore, in our base algorithm, we use the subsequence inequalities (3–31) and (3–32) described in Section 3.2.4.2. We also use Engel's heuristic (Engel, 2005), which executes in well under one CPU second for each instance and generates a solution having minimum beam-on-time, to (i) obtain an initial upper bound and (ii) compute the upper bounds R_ℓ $(\ell = 1, \ldots, L)$.

3.3.3 Comparison with Langer et al. (2001) Model

Our first experiment compares our base algorithm that minimizes the total treatment time to that of Langer et al. (2001) and to the modification of their model as described in Appendix A. We choose randomly generated test instances of various dimensions to identify the problem sizes that can be solved by each algorithm, as well as four of the smallest clinical instances to compare the effectiveness of the algorithms on clinical instances. We imposed a one-hour time limit past which we halted the execution of an algorithm. For these experiments we disabled the use of Engel's heuristic as an initial heuristic to test the ability of these models to efficiently find good-quality upper bounds.

Table 3-2 summarizes the results of these three algorithms in terms of the execution time, the best upper and lower bounds found within the time limit, and the optimality gap (calculated as the difference between the upper and lower bound as a percentage of the upper bound). Our decomposition algorithm can solve all 15 instances in this data set within a few seconds, whereas only six instances can be solved to optimality within an hour by either integer programming formulation. We conclude that, even though the integer programming formulation given in (Langer et al., 2001) can solve small instances to optimality, it cannot be used to solve clinical problem instances to optimality within practical computation time limits.

Table 3-2. Comparison of our base algorithm with Langer et al. (2001) model

				Two stage		Langer				Modified Langer			
Name	m	n	L	CPU	Optimal	CPU	UB	LB	Gap	CPU	UB	LB	Gap
test3x3	3	3	8	0.1	29	1.6	29	29.00	0.0%	0.9	29	29.00	0.0%
test3x4	3	4	8	0.1	37	5.2	37	37.00	0.0%	1.6	37	37.00	0.0%
test4x4	4	4	8	0.1	36	30.4	36	36.00	0.0%	10.7	36	36.00	0.0%
test5x5a	5	5	10	0.2	45	2069.6	45	45.00	0.0%	86.4	45	45.00	0.0%
test5x5b	5	5	15	0.2	50	198.2	50	50.00	0.0%	92.6	50	50.00	0.0%
test5x6a	5	6	10	0.2	55	3600	61	33.53	45.0%	3600	55	40.95	25.5%
test5x6b	5	6	18	0.4	71	3600	84	51.58	38.6%	3600	77	58.67	23.8%
test6x6a	6	6	13	0.3	55	3600	55	45.63	17.0%	3600	55	48.00	12.7%
test6x6b	6	6	13	0.3	52	3600	57	43.82	23.1%	3600	57	50.00	12.3%
test6x7a	7	6	10	0.2	45	690.0	45	45.00	0.0%	435.1	45	45.00	0.0%
test6x7b	6	7	15	0.4	74	3600	94	35.69	62.0%	3600	80	47.88	40.1%
c1b1	15	14	20	1.3	111	3600	336	48.58	85.5%	3600	273	42.00	84.6%
c1b2	11	15	20	0.8	104	3600	280	38.26	86.3%	3600	132	39.55	70.0%
c1b5	11	15	20	3.1	104	3600	280	46.20	83.5%	3600	140	49.29	64.8%
c5b4	14	16	20	2.5	124	3600	360	34.00	90.6%	3600	360	39.11	89.1%

3.3.4 Random Problem Instances

For our next experiment, we first solved each of the 20 × 20 random problem instances in our data set to optimality for the problems of (i) minimizing total treatment time ("Total Time"), (ii) minimizing the number of apertures while constraining the beam-on-time to be minimal ("Lexicographic"), and (iii) minimizing the number of apertures ("# Apertures"). We also implemented three heuristic algorithms proposed by Siochi (2007), Engel (2005), and Xia and Verhey (1998), which we executed on the same data set. (The results we present from Siochi (2007) refer to the Variable Depth Recursion (VDR) algorithm without tongue-and-groove constraints, using the parameters recommended in the paper. We discuss the effect of including tongue-and-groove considerations in the algorithm below.)

Figure 3-2 summarizes the total treatment times associated with the solutions generated by the six algorithms we tested. Each algorithm is represented by a curve that depicts quality of the solutions obtained by the corresponding algorithm. For each value T of total treatment time on the horizontal axis, each curve plots the number of problem instances for which the corresponding algorithm was able to find a solution having total treatment time no more than T. For instance, Figure 3-2 shows that Siochi's heuristic found a solution with a total treatment time of at most 175 time units in 5% of the problem instances, while an optimal solution (represented by "Total Time") has

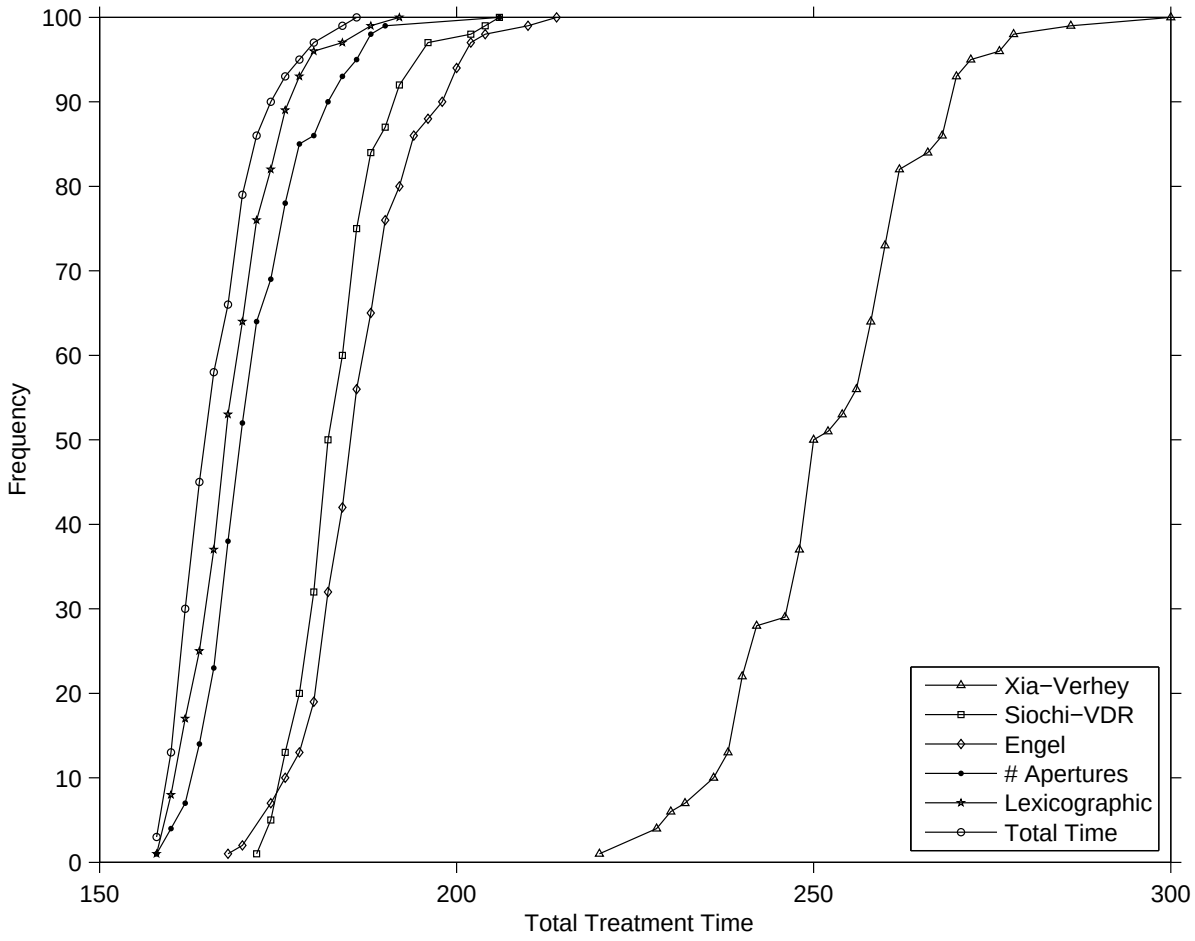


Figure 3-2. Comparison of total treatment times on random data

the same quality level in 97% of the problem instances. We observe that all three exact algorithms find solutions having similar treatment times. Solution qualities generated by the Engel and Siochi heuristics are similar, with the Siochi heuristic being slightly better. A comparison of the heuristic solutions with optimal solutions reveals that average optimality gaps for Siochi, Engel and Xia-Verhey heuristics are 10.1%, 12.0%, and 51.5%, respectively.

Figure 3-3 compares the algorithms with respect to the number of apertures used in their respective solutions. We note that our algorithm that minimizes total treatment time ("Total Time") finds a solution that also minimizes the number of apertures for most problem instances. As expected, lexicographic minimization of the two objective functions

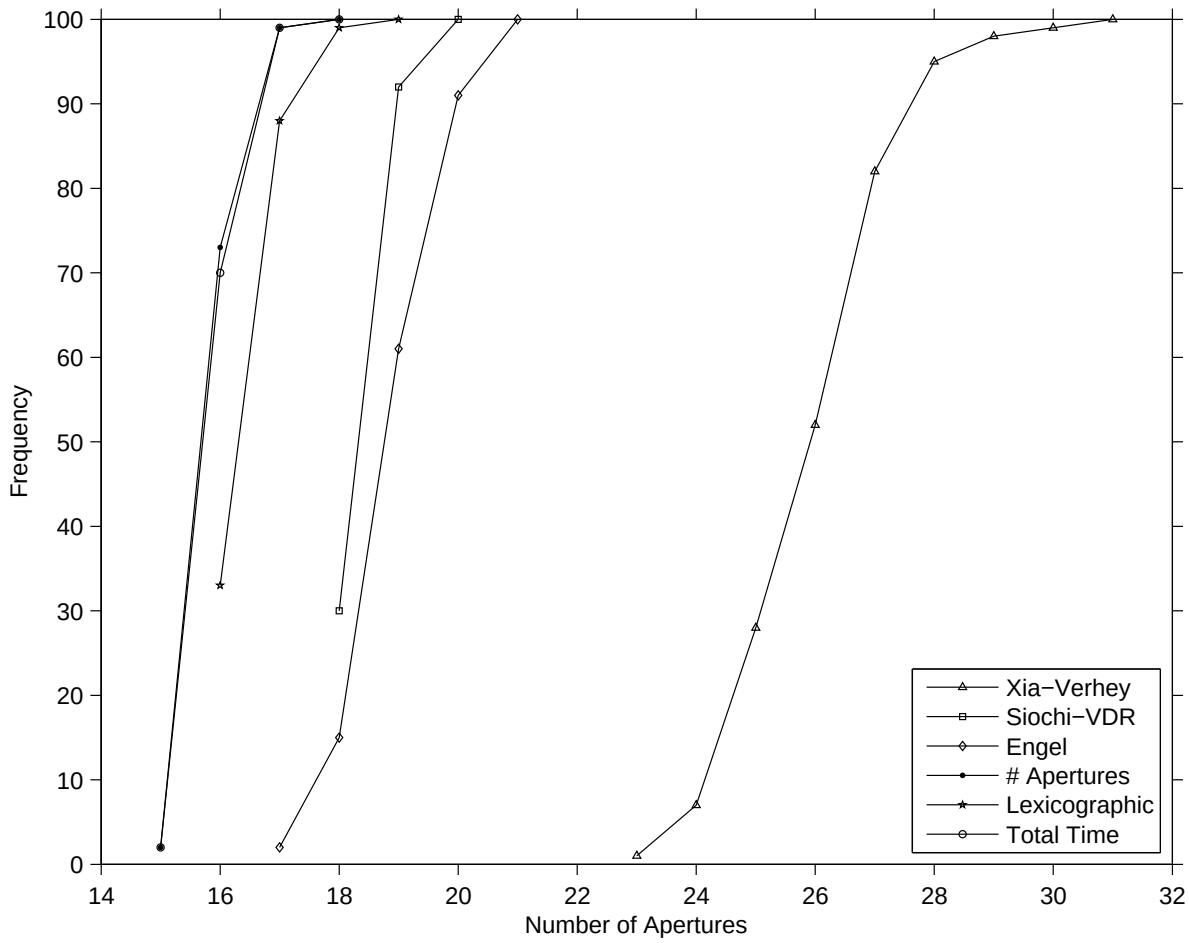


Figure 3-3. Comparison of the number of apertures on random data

results in an increased number of apertures. For this objective the "# Apertures" algorithm finds optimal solutions. Average optimality gaps for the heuristics of Siochi, Engel, and Xia-Verhey are 15.6%, 18.9%, and 62.3%, respectively.

We analyze the beam-on-time values of the solutions generated by each algorithm in Figure 3-4. Since both Engel's heuristic and our "Lexicographic" algorithm find optimal solutions having minimum beam-on-time, their curves overlap. We observe that the Siochi heuristic and our "Total Time" algorithm tend to generate solutions having small beam-on-time values, but the solutions generated by our "# Apertures" algorithm, and by the Xia-Verhey heuristic have higher beam-on-time values. We calculated the average optimality gaps for the latter two algorithms as 12.6% and 32.1%, respectively.

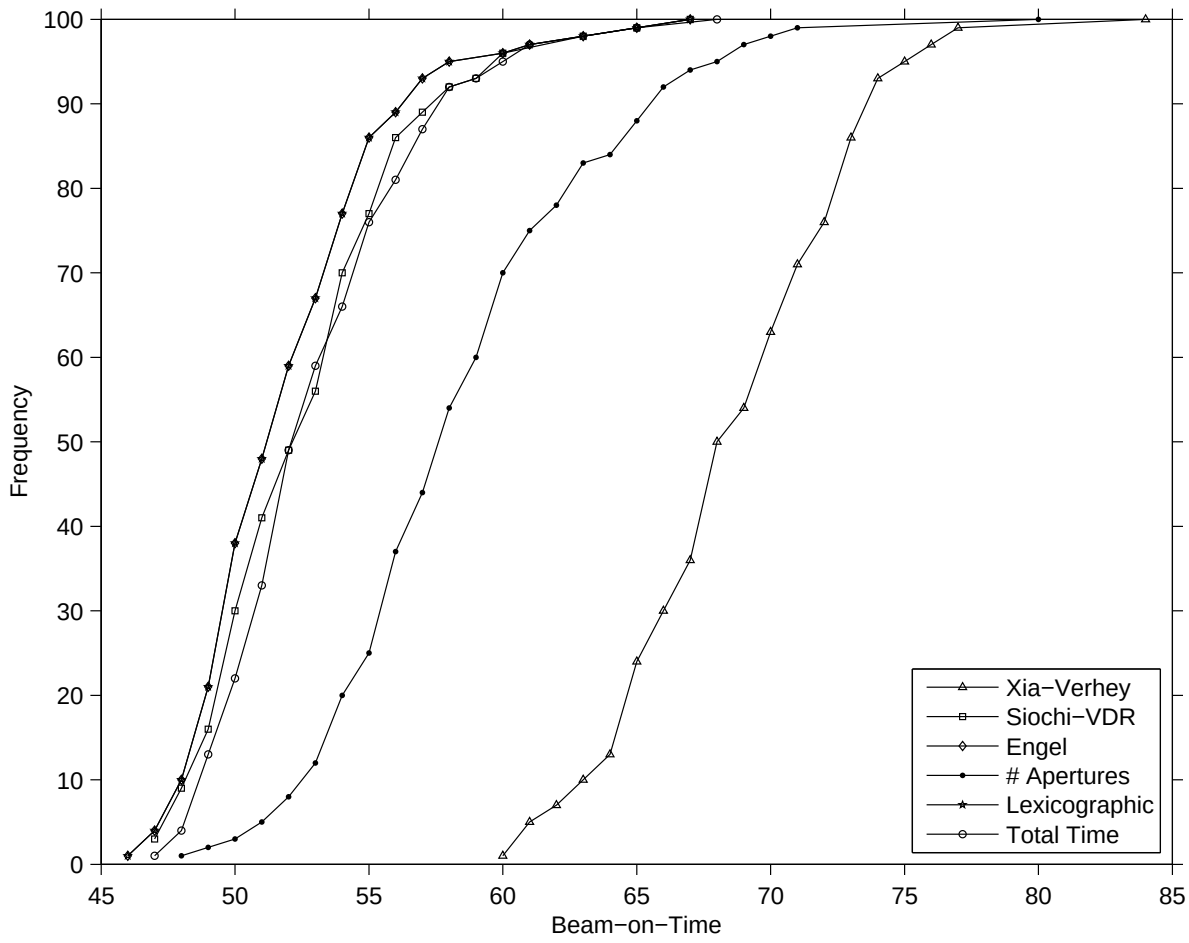


Figure 3-4. Comparison of beam-on-time values on random data

The final measure of solution quality that we consider is TGI, which is a measure of the tongue-and-groove effect given by (3–33). Figure 3-5 reveals that the solutions obtained by all three variants of our decomposition algorithm have significantly lower TGI values than the heuristic procedures. This result implies that, even though our TGI-reduction algorithm described in Section 3.2.5 does not guarantee a minimum TGI, it is highly effective in finding solutions with TGI values superior to the other heuristic approaches. To estimate optimality gaps for the heuristics we compare heuristic solutions with the solutions generated by our "Lexicographic" algorithm, which provides the best TGI among all methods mentioned above. We note that average gaps for Siochi, Engel and Xia-Verhey heuristics are 162.1%, 164.4%, and 205.4%, respectively. We also note

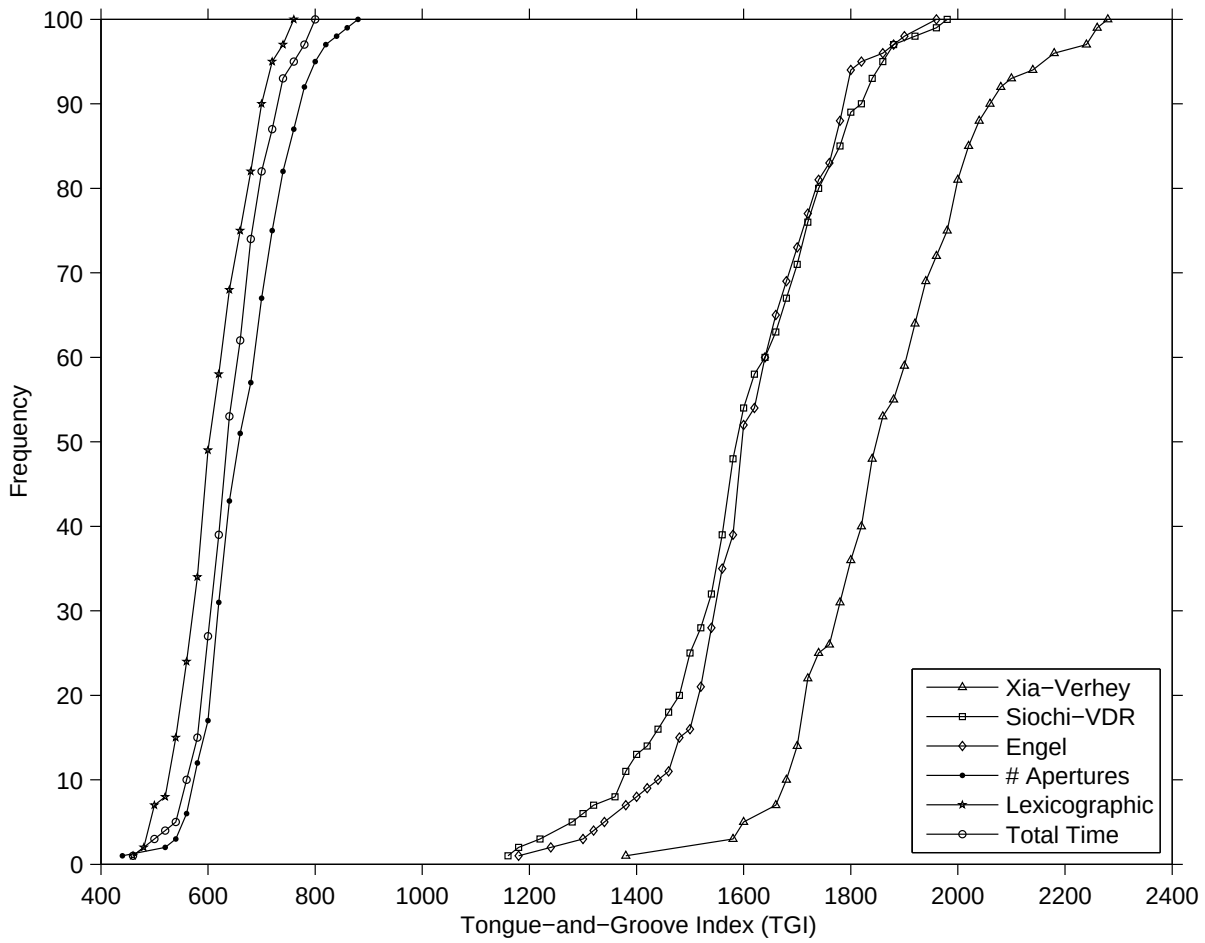


Figure 3-5. Comparison of TGI values on random data

that these heuristics do not attempt to minimize TGI, and it might be possible to modify them to obtain solutions with lower TGI values. It is interesting to note that a variant of Siochi's algorithm (Siochi, 2007) is capable of completely eliminating TGI at the expense of creating additional apertures. This variant is reported to increase the number of apertures by 10% to 30% relative to the variant that does not remove TGI (Siochi, 2007).

Finally, the Engel and Xia-Verhey heuristics took less than one second of CPU time in all instances we tested. The average CPU time for Siochi's heuristic, "Total Time" algorithm, "# Apertures" algorithm, and "Lexicographic" algorithm were 31.5, 963.1, 414.8, and 421.4 seconds, respectively. We note that all variants of our two-stage algorithm showed a "heavy-tail" behavior, where about 80% of the problem instances were solved to

optimality in less than the average solution time. For instance, using the "# Apertures" algorithm, we were able to solve 40 instances within one minute, 58 within two minutes, 81 within 414.8 seconds (the average solution time for this algorithm), 90 within 15 minutes, and all but three instances were solved within an hour. The remaining three instances were solved within three hours.

3.3.5 Clinical Problem Instances

Recall from Section 3.1 that in clinical practice, we can deliver each fluence map using a decomposition into either row-convex or column-convex apertures, where the latter requires rotation of the MLC head. Our final set of experiments compares the algorithms on clinical problem instances in our data set, allowing for MLC head rotation.

We first show the results of applying our decomposition algorithm to decompose each of the 25 clinical fluence maps into row-convex apertures, and column-convex apertures, where the latter is achieved by applying our algorithm to the transpose of each fluence map. Table 3-3 reports the performance of our algorithm when the objective function is set to minimize total treatment time, and displays the number of apertures ("nAper"), beam-on-time ("BOT"), total treatment time ("Time"), tongue-and-groove index ("TGI"), and CPU time used ("CPU") for the algorithm.

Our algorithm finds an optimal solution to several instances within a few seconds while four instances take more than 10 minutes of CPU time to be solved to optimality. Comparing the solutions obtained for row-convex and column-convex decompositions, we observe that rotating the MLC head is most beneficial (in terms of treatment time) for instances in which the number of rows is much smaller than the number of columns. These benefits are most apparent on instances c4b2 and c4b5, where rotating the MLC head can result in more than 50% reduction in total treatment time. We also note that several problem instances require much less computational time to solve for a column-convex decomposition compared to a row-convex decomposition.

Table 3-3. Effect of rotating the MLC head

	Row-Convex					Column-Convex				
Name	nAper	BOT	Time	TGI	CPU	nAper	BOT	Time	TGI	CPU
c1b1	10	41	111	102	1.1	11	38	115	50	5.5
c1b2	10	34	104	80	0.8	8	23	79	14	0.7
c1b3	11	31	108	97	11.4	9	28	91	59	1.0
c1b4	11	33	110	74	37.0	11	37	114	146	7.0
c1b5	10	34	104	133	4.3	8	32	88	49	1.2
c2b1	14	34	132	134	26.5	12	30	114	187	11.5
c2b2	13	41	132	159	20.1	11	33	110	192	8.0
c2b3	13	49	140	245	14.7	11	28	105	151	3.1
c2b4	14	51	149	316	87.3	12	34	118	148	8.3
c2b5	13	41	132	217	395.6	10	27	97	120	2.0
c3b1	13	41	132	323	310.0	14	40	138	254	23.0
c3b2	14	46	144	320	4759.8	8	23	79	86	1.1
c3b3	13	49	140	533	10373.9	12	40	124	360	18.6
c3b4	12	44	128	481	524.9	12	40	124	327	428.2
c3b5	13	34	125	133	3.3	9	27	90	75	2.6
c4b1	16	40	152	216	34.9	12	46	130	244	10.6
c4b2	16	69	181	450	20901.0	9	27	90	149	15.8
c4b3	14	41	139	130	44.7	10	32	102	129	3.3
c4b4	14	44	142	246	164.3	10	27	97	163	8.0
c4b5	17	76	195	470	14511.4	9	24	87	48	4.0
c5b1	10	26	96	68	0.5	10	35	105	41	0.5
c5b2	12	41	125	59	14.3	8	25	81	27	0.6
c5b3	10	34	104	155	3.1	9	23	86	42	1.0
c5b4	12	40	124	105	2.2	10	32	102	87	4.3
c5b5	12	46	130	151	51.9	8	31	87	17	0.8

Table 3-4. Computational results for our base algorithm

	Total Time				# Apertures				Lexicographic			
Name	nAper	BOT	TGI	CPU	nAper	BOT	TGI	CPU	nAper	BOT	TGI	CPU
c1b1	10	41	102	4.7	10	41	102	2.3	11	38	50	4.8
c1b2	8	23	14	1.1	8	23	14	1.1	8	23	14	1.1
c1b3	9	28	59	3.0	9	28	59	4.5	9	28	59	4.5
c1b4	11	33	74	41.2	11	37	128	27.1	11	33	74	12.2
c1b5	8	32	49	2.1	8	34	56	1.3	9	26	9	1.9
c2b1	12	30	187	15.6	12	30	187	14.9	12	30	187	14.4
c2b2	11	33	192	10.8	11	38	161	6.9	11	33	146	7.8
c2b3	11	28	149	8.9	11	28	113	9.9	11	28	197	10.8
c2b4	12	34	148	16.8	12	34	148	16.8	12	34	148	17.1
c2b5	10	27	120	6.1	10	31	155	6.2	10	27	120	6.2
c3b1	13	41	323	315.0	12	51	521	62.1	13	41	325	31.4
c3b2	8	23	86	4.4	8	26	87	4.5	8	23	62	5.6
c3b3	12	40	360	27.4	12	40	360	894.7	12	40	365	20.1
c3b4	12	40	327	442.2	12	46	284	548.8	13	38	928	55.1
c3b5	9	27	75	5.6	9	27	75	5.4	9	27	75	5.7
c4b1	12	46	244	16.8	12	46	227	10.6	12	46	227	11.3
c4b2	9	27	149	45.5	9	32	150	56.2	9	27	135	35.0
c4b3	10	32	129	15.7	10	34	108	14.9	10	32	129	15.6
c4b4	10	27	163	32.0	10	28	112	32.6	11	26	72	29.9
c4b5	9	24	48	27.8	9	24	48	27.7	9	24	48	27.0
c5b1	10	26	68	1.2	10	26	68	1.2	10	26	68	1.2
c5b2	8	25	27	1.1	8	25	27	1.0	9	23	8	1.1
c5b3	9	23	42	3.6	9	24	45	3.2	9	23	83	3.1
c5b4	10	32	87	5.8	10	41	101	2.7	10	32	87	2.8
c5b5	8	31	17	1.4	8	33	16	1.2	8	31	71	1.1

Motivated by this observation, we modify our algorithm to directly solve for the best orientation by using obtained upper and lower bounds to quickly prove whether rotating the MLC head is beneficial. Assume that we have lower and upper bounds for the row-convex and column-convex problems, and suppose that the lower bound of the row-convex problem is greater than the upper bound of the column-convex problem. In this case, we can conclude that an optimal solution minimizing total treatment time for the given fluence map must be a column-convex decomposition. We use this argument to solve one of the problems, and then use the bound information to avoid having to solve the other one to optimality. We pick the first problem to solve by selecting one having the least initial lower bound, breaking ties if applicable by choosing the problem for which $n < m$, since the subproblems tend to solve faster for smaller values of n. Table 3-4 shows the nAper, BOT, TGI, and CPU metrics obtained from our algorithm enhanced with the above bounding scheme, corresponding to the "Total Time," "# Apertures," and "Lexicographic" objectives. Observe that all 25 instances, under any metric, terminate in under 15 minutes of CPU time with a solution that is optimal with respect to the corresponding objective, and all instances are solved to optimality within a minute using the "Lexicographic" algorithm.

Recall that the "BOT" column in "Lexicographic" reports the minimum achievable beam-on-time, and the "nAper" column under the objective "# Apertures" reports the minimum number of apertures needed to decompose each instance. Perhaps surprisingly, in comparing these values with the results of "Total Time," we observe that there exists a solution that minimizes both the number of shapes and the beam-on-time simultaneously in 19 of the 25 instances.

Finally, we analyze performance of the three heuristics on clinical data, where we execute each heuristic on each problem instance and its transpose (corresponding to row-convex and column-convex decompositions), and pick the solution yielding the smallest treatment time. Table 3-5 shows the number of apertures, beam-on-time, TGI

Table 3-5. Comparison of heuristic algorithms on clinical data

	Siochi				Engel				Xia-Verhey			
Name	nAper	BOT	TGI	CPU	nAper	BOT	TGI	CPU	nAper	BOT	TGI	CPU
c1b1	11	38	245	14.0	12	38	261	< 1	13	40	219	< 1
c1b2	8	23	109	3.0	8	23	127	< 1	10	32	133	< 1
c1b3	9	28	213	4.0	10	28	192	< 1	12	34	198	< 1
c1b4	12	34	306	9.5	11	37	398	< 1	14	42	355	< 1
c1b5	9	26	103	3.6	9	26	175	< 1	12	35	124	< 1
c2b1	12	30	652	11.2	12	30	738	< 1	15	45	635	< 1
c2b2	12	33	395	17.8	12	33	464	< 1	15	45	460	< 1
c2b3	12	28	625	34.8	12	28	429	< 1	15	43	459	< 1
c2b4	12	34	628	43.2	12	34	723	< 1	18	56	417	< 1
c2b5	11	27	463	15.3	11	27	465	< 1	14	41	375	< 1
c3b1	14	43	828	36.3	15	40	1054	< 1	17	55	765	< 1
c3b2	9	23	143	11.1	9	23	127	< 1	12	36	289	< 1
c3b3	14	40	1316	40.8	14	40	869	< 1	19	60	1038	< 1
c3b4	13	48	678	33.3	14	38	765	< 1	17	55	553	< 1
c3b5	9	28	263	7.0	9	27	325	< 1	13	45	261	< 1
c4b1	13	46	617	29.4	14	46	625	< 1	18	62	531	< 1
c4b2	10	29	295	73.8	10	27	466	< 1	14	44	350	< 1
c4b3	11	32	339	19.4	11	32	365	< 1	14	48	428	< 1
c4b4	11	26	489	13.5	11	26	540	< 1	15	46	424	< 1
c4b5	9	24	236	89.8	9	24	328	< 1	15	44	328	< 1
c5b1	11	26	188	4.6	12	26	176	< 1	12	38	185	< 1
c5b2	9	23	129	6.9	9	23	100	< 1	10	33	145	< 1
c5b3	9	26	201	5.1	10	23	293	< 1	12	32	189	< 1
c5b4	11	32	218	11.2	11	32	322	< 1	13	46	243	< 1
c5b5	8	32	217	7.2	9	31	211	< 1	11	35	138	< 1

metrics for each solution as well as the CPU time spent by each heuristic. Comparison with the "Total Time" columns in Table 3-4 reveals that even though the heuristics consistently generated high-quality solutions, the Siochi and Engel heuristics were able to find an optimal solution in only five problem instances, and Xia-Verhey heuristic could not find an optimal solution to any instance.

CHAPTER 4
RECTANGULAR MATRIX DECOMPOSITION PROBLEM

4.1 Introduction and Literature Survey

Over the past decade, Intensity Modulated Radiation Therapy (IMRT) has developed into the most successful external-beam radiation therapy delivery technique for many forms of cancer. This is due to its ability to deliver highly complex dose distributions to cancer patients that enable the eradication of cancerous cells while limiting damage to nearby healthy organs and tissues. Patients treated with IMRT therefore often experience a higher chance of cure, suffer from fewer side effects of the treatment, or both. In this chapter, we study an optimization problem that is related to the efficient clinical implementation of IMRT using a simpler technology than currently used, which, if successful, will reduce the cost as well as the complexity of delivering IMRT and thereby make such superior treatments accessible to significantly more patients worldwide.

External-beam radiation therapy is delivered from multiple angles by a device that can rotate around a patient. The use of multiple (typically 3–9) angles is one of the tools that allow for the treatment of deep-seated tumors while limiting the radiation dose to surrounding functioning organs. Conventional conformal radiation therapy then further uses blocks and wedges to shape the beams (see, e.g., Lim (2002) and Lim et al. (2004, 2007)). IMRT is a more powerful therapy that instead modulates beam intensity. The most common technique for achieving this modulation is to dynamically shape beams with the help of a multileaf collimator (MLC) system. Such systems can dynamically form many complex apertures by independently moving leaf pairs that block part of the radiation beam. Unfortunately, MLC systems are very costly and technologically advanced, and are therefore difficult and expensive to operate and maintain. Moreover, MLC systems are currently only available for use with a so-called linear accelerator that generates high-energy photon beams for treatment. However, the use of radioactive ^{60}Co (Cobalt) sources for radiation therapy is still ubiquitous in many parts of the world and is

poised to experience a revival in the United States and Europe through the Renaissance^{TM} device that is under development by ViewRay, Inc. based in Cleveland, Ohio. Without a MLC, IMRT delivery may be achieved through the use of compensators: high-density blocks that control the intensity profile of a radiation beam. Such blocks are custom-made for each individual patient, which makes compensator-based IMRT not only labor and storage space intensive, but it also makes the actual treatment very time-consuming due to the fact that therapists must enter the treatment room to place each individual compensator. In addition, compensators have several undesirable properties that make it difficult to perform accurate dose calculations, thereby reducing the advantages of IMRT (see, e.g., Earl et al. (2007)). Recently, researchers have begun to explore the clinical feasibility of delivering IMRT using conventional jaws that are already integrated into radiation delivery devices and can create apertures that are rectangular in shape (see, e.g., Earl et al. (2007), Kim et al. (2007), and Men et al. (2007)). Successful application of this much simpler delivery technique depends critically on the ability to *efficiently* deliver high-quality treatment plans. We therefore develop and test new optimization approaches to minimize the treatment time required for a particular treatment plan using rectangular apertures only.

Solving a so-called fluence map optimization problem yields an optimal IMRT treatment plan that resolves different, and conflicting, clinical measures of treatment plan quality related to tumor control and side effects (see, e.g., Shepard et al. (1999) for a review; Lee et al. (2000a, 2003) for mixed-integer programming approaches; Romeijn et al. (2006) for convex programming models; and Hamacher and Küfer (2002) and Küfer et al. (2003) for a multicriteria approach). A treatment plan then consists of a collection of nonnegative intensity matrices, often referred to as fluence maps, one corresponding to each beam angle. To limit treatment time, each of these matrices is then expressed as a multiple of an integral fluence map in which the maximum element is on the order of 10–20. To allow delivery of the treatment plan, each of these fluence maps should be

decomposed into a number of apertures and corresponding intensities, where the collection of apertures that may be used depends on the delivery equipment. For MLC delivery this problem is called the leaf sequencing problem and is very widely studied; for examples, we refer to Ahuja and Hamacher (2005), Boland et al. (2004), Kamath et al. (2003), Engel (2005), Kalinowski (2005a), and Taşkın et al. (2009b). (Note that integrated approaches to fluence map optimization, also referred to as aperture modulation, have been proposed as well; we refer to, e.g., Preciado-Walters et al. (2004), Romeijn et al. (2005), and Men et al. (2007).)

The problem that we study is the decomposition of an integral fluence map into rectangular apertures and corresponding intensities. While Dai and Hu (1999) proposed a straightforward heuristic for a variant of this decomposition problem, we develop the first computationally viable optimization approach to this problem. In Section 4.2 we consider the core problem of decomposing an (integral) fluence map while minimizing the number of rectangular apertures. In Section 4.3 we then extend our models to the problems of (i) minimizing total treatment time (as measured by the sum of the required aperture setup times and the beam-on-time, i.e., the actual time that radiation is being delivered); and (ii) minimizing the number of apertures subject to beam-on-time being minimal. Finally, Section 4.4 discusses our computational results on a collection of clinical fluence maps.

4.2 A Mixed-Integer Programming Approach

We begin in Section 4.2.1 by formally describing the optimization model under investigation and modeling it with a mixed-integer programming formulation. We next describe several classes of valid inequalities in Section 4.2.2. Finally, we discuss methods for partitioning the input matrix in Section 4.2.3, which leads to effective lower and upper bounding techniques.

4.2.1 Model Development

In this section, we discuss an integer programming approach to decomposing a fluence map into a minimum number of rectangular apertures and corresponding intensities. We

denote the fluence map to be delivered by a matrix $B \in \mathbb{N}^{m \times n}$, where the element at row i and column j, (i, j), corresponds to a bixel with required intensity b_{ij}. We call a bixel having an intensity requirement of zero a *zero-bixel*. We also define a *nonzero-bixel* analogously. Figure 4-1 shows an example fluence map, which we use throughout this chapter.

2	3	**0**	8	2	4	2
2	1	**0**	5	1	2	1
3	**0**	**0**	5	**0**	**0**	3
5	**0**	2	8	6	**0**	3
0	8	14	10	9	**0**	3
5	8	20	7	1	**0**	4
5	9	5	4	**0**	**0**	3

Figure 4-1. Example fluence map

Let R be the set of all $O(n^2m^2)$ possible rectangular apertures (i.e., submatrices of B having contiguous rows and columns) that can be used to decompose B, excluding those that contain a zero-bixel. For each rectangle $r \in R$ we define a continuous variable x_r that represents the intensity assigned to rectangle r, and a binary variable y_r that equals 1 if rectangle r is used in decomposing B (i.e., if $x_r > 0$), and equals 0 otherwise. Let C_r be the set of bixels that is exposed by rectangle r. We define $M_r = \min_{(i,j) \in C_r}\{b_{ij}\}$ to be the minimum intensity requirement among the bixels covered by rectangle r. Furthermore, we denote the set of rectangles that cover bixel (i, j) by $R(i, j)$. Given these definitions, we can formulate the problem as follows:

$$\textbf{IPR: Minimize } \sum_{r \in R} y_r \tag{4–1}$$

$$\text{subject to: } \sum_{r \in R(i,j)} x_r = b_{ij} \quad \forall i = 1, \ldots, m,\ j = 1, \ldots, n \tag{4–2}$$

$$x_r \leq M_r y_r \quad \forall r \in R \tag{4–3}$$

$$x_r \geq 0,\ y_r \text{ binary} \quad \forall r \in R. \tag{4–4}$$

The objective function (4–1) minimizes the number of rectangles used in the decomposition. Constraints (4–2) guarantee that each bixel receives exactly the required dose. Constraints (4–3) enforce the condition that x_r cannot be positive unless $y_r = 1$. Finally, (4–4) states bounds and logical restrictions on the variables. Note that the objective (4–1) guarantees that $y_r = 0$ when $x_r = 0$ in any optimal solution of IPR.

Formulation IPR contains two variables and a constraint for each rectangle, resulting in a large-scale mixed-integer program for problem instances of clinically relevant sizes. Furthermore, the M_r-terms in constraints (4–3) lead to a weak linear programming relaxation; with no valid inequalities or branching yet performed on the problem, we have that $y_r = x_r/M_r$ at optimality to the linear programming relaxation of IPR. An alternative formulation that does not require M_r-terms employs a decomposition method. Recall that we investigated the problem of decomposing an integer matrix into "consecutive-ones" matrices in Chapter 3, where in each decomposed matrix all nonzero values take the same value and appear consecutively on each row. Our computational results showed that solvability of the problem is significantly improved by applying a bi-level optimization algorithm. A similar approach for the problem we consider in this chapter would formulate a master problem as:

$$\textbf{MP: Minimize } \sum_{r \in R} y_r \tag{4–5}$$

$$\text{subject to: } \mathbf{y} \text{ corresponds to a feasible decomposition} \tag{4–6}$$

$$y_r \text{ binary} \quad \forall r \in R, \tag{4–7}$$

where we address the form of (4–6) in the sequel. Given a vector $\hat{\mathbf{y}}$, we can check whether constraint (4–6) is satisfied by solving the following linear program:

$$\textbf{SP}(\hat{\mathbf{y}})\text{: Minimize } 0 \tag{4–8}$$

$$\text{subject to: } \sum_{r \in R(i,j)} x_r = b_{ij} \quad \forall i = 1, \ldots, m,\ j = 1, \ldots, n \tag{4–9}$$

$$x_r \leq M_r \hat{y}_r \quad \forall r \in R \tag{4–10}$$

$$x_r \geq 0 \quad \forall r \in R. \tag{4–11}$$

Associating variables α_{ij} with (4–9), and β_r with (4–10), we obtain the dual formulation:

$$\textbf{DSP}(\hat{\mathbf{y}}): \text{ Maximize } \sum_{i=1}^{m}\sum_{j=1}^{n} b_{ij}\alpha_{ij} + \sum_{r\in R} M_r \hat{y}_r \beta_r \tag{4–12}$$

$$\text{subject to: } \sum_{(i,j)\in C_r} \alpha_{ij} + \beta_r \geq 0 \quad \forall r \in R \tag{4–13}$$

$$\alpha_{ij} \text{ unrestricted} \quad \forall i = 1, \ldots, m,\ j = 1, \ldots, n \tag{4–14}$$

$$\beta_r \leq 0 \quad \forall r \in R. \tag{4–15}$$

Our Benders decomposition strategy first solves MP, which yields $\hat{\mathbf{y}}$. If SP($\hat{\mathbf{y}}$) is feasible, then $\hat{\mathbf{y}}$ corresponds to a feasible decomposition and is optimal. Else, DSP($\hat{\mathbf{y}}$) is unbounded (since the trivial all-zero solution guarantees its feasibility). Let ($\hat{\boldsymbol{\alpha}}$, $\hat{\boldsymbol{\beta}}$) be an extreme dual ray of DSP($\hat{\mathbf{y}}$) such that $\sum_{i=1}^{m}\sum_{j=1}^{n} b_{ij}\hat{\alpha}_{ij} + \sum_{r\in R} M_r \hat{y}_r \hat{\beta}_r > 0$. Then, all $\mathbf{y}$-vectors that are feasible with respect to (4–6) must satisfy

$$\sum_{i=1}^{m}\sum_{j=1}^{n} b_{ij}\hat{\alpha}_{ij} + \sum_{r\in R} (M_r \hat{\beta}_r) y_r \leq 0. \tag{4–16}$$

We add (4–16) in a cutting plane fashion as necessary.

Remark 6. *Even though the number of rectangles that can be used in partitioning the input matrix B is $O(n^2m^2)$, we observe that optimal solutions typically use only a small percentage of the total number of rectangles. This observation suggests that another way to overcome the dimensional complexity associated with solving IPR is to apply a column generation approach. In this approach, we start with a feasible set of columns and rows corresponding to a subset of rectangles and generate additional columns and rows as necessary within a branch-cut-price (BCP) algorithm. Even though this approach requires the solution of much smaller linear programming relaxations, several features of the branch-and-cut algorithm such as preprocessing and automatic cutting plane*

generation are not applicable. As a result, our implementation of the BCP approach was not computationally competitive with the other algorithms we presented, and further details are therefore omitted.

4.2.2 Valid Inequalities

In this section we discuss several valid inequalities and optimality conditions for our problem. All inequalities that we describe in this section are applicable to both the integer programming formulation and the master problem of the Benders decomposition approach we described in Section 4.2.1.

4.2.2.1 Adjacent rectangles

We call two non-overlapping rectangles r_1 and r_2 *adjacent* if either of the following conditions is satisfied:

(a) r_1 and r_2 cover an identical range of columns, with r_1 having bottom row i and r_2 having top row $i+1$, or

(b) r_1 and r_2 cover an identical range of rows, with r_1 having right-most column j and r_2 having left-most column $j+1$.

We observe that there exists an optimal solution in which no two adjacent rectangles are used in the decomposition. To see this, assume that adjacent rectangles r_1 and r_2 have intensities x_{r_1} and x_{r_2}, respectively, where $x_{r_1} \leq x_{r_2}$ without loss of generality. In this case, an alternative optimal solution can be constructed by extending r_1 into r_2. Specifically, let r' be the rectangle for which $C_{r'} = C_{r_1} \cup C_{r_2}$. An alternative optimal solution that does not contain any adjacent rectangles uses r_2 having intensity $x_{r_2} - x_{r_1}$, and r' having intensity x_{r_1}. This dominance criterion can be written as:

$$y_{r_1} + y_{r_2} \leq 1 \quad \forall \text{ adjacent rectangles } r_1, r_2, \tag{4–17}$$

which states that no pair of adjacent rectangles can be selected in an optimal solution.

4.2.2.2 Bounding box inequalities

We first observe that intensity requirements of adjacent bixels can be used to derive certain necessary conditions that any feasible decomposition of a matrix needs to satisfy.

We say that a rectangle *starts* at bixel (i, j) if the upper-left corner of the rectangle is located at (i, j). Consider the bixel $(5, 3)$ marked with dark gray in Figure 4-2. Since $b_{43} = 2$, the total intensity delivered to $(5, 3)$ by all rectangles that start in rows $i = 1, \ldots, 4$ cannot exceed 2. However, $b_{53} = 14 > 2$, and hence at least one rectangle that starts in row 5 is required to cover bixel $(5, 3)$. Similarly, $b_{53} > b_{52}$ implies that at least one rectangle that starts in column 3 is required to cover the same bixel. These results can be strengthened by considering both $(4, 3)$ and $(5, 2)$ simultaneously. Since $b_{53} > b_{43} + b_{52}$, we conclude that at least one rectangle that starts at bixel $(5, 3)$ is required in any feasible decomposition of the fluence map. In general, a rectangle must start at (i, j) if $b_{ij} > b_{(i-1)j} + b_{i(j-1)}$ is satisfied. Figure 4-3 illustrates a similar idea, where we

2	3	**0**	8	2	4	2
2	1	**0**	5	1	2	1
3	**0**	**0**	5	**0**	**0**	3
5	**0**	2	8	6	**0**	3
0	8	14	10	9	**0**	3
5	8	20	7	1	**0**	4
5	9	5	4	**0**	**0**	3

Figure 4-2. Example start index

compare the intensity requirement of bixel $(6, 4)$ with the bixel below it, and the one on its right. Using arguments similar to the ones regarding starting indices, we conclude that a rectangle must *end* (i.e., have a lower-right corner) at $(6, 4)$ since $b_{64} > b_{74} + b_{65}$.

2	3	**0**	8	2	4	2
2	1	**0**	5	1	2	1
3	**0**	**0**	5	**0**	**0**	3
5	**0**	2	8	6	**0**	3
0	8	14	10	9	**0**	3
5	8	20	7	1	**0**	4
5	9	5	4	**0**	**0**	3

Figure 4-3. Example end index

Starting and ending index conditions can be generalized further as follows. Assume that there exist integers $u \in [0, i-1]$, $d \in [i+1, m+1]$, $l \in [0, j-1]$, and $r \in [j+1, n+1]$

so that $b_{ij} > b_{il} + b_{uj} + b_{ir} + b_{dj}$, where we define $b_{i0} = b_{0j} = b_{m+1,j} = b_{i,n+1} = 0$ for $i \in \{0, \ldots, m+1\}, j \in \{0, \ldots, n+1\}$. In this case, we say that (l, u, r, d) is a *bounding box* for bixel (i, j). Figure 4-4 illustrates a bounding box for bixel $(6, 3)$ (marked in dark gray), which corresponds to $(l, u, r, d) = (2, 4, 5, 7)$. The four bixels that represent the borders of a bounding box are marked in light gray. We note that any rectangle that

2	3	**0**	8	2	4	2
2	1	**0**	5	1	2	1
3	**0**	**0**	5	**0**	**0**	3
5	**0**	2	8	6	**0**	3
0	8	14	10	9	**0**	3
5	8	20	7	1	**0**	4
5	9	5	4	**0**	**0**	3

Figure 4-4. Example bounding box

contains bixel $(6, 3)$, and does not start inside the bounding box (at (5,3) or (6,3)) or end inside the bounding box (at (6,3) or (6,4)), has to contain at least one of the four bixels on the border. Therefore, the sum of intensities of those rectangles is bounded by the total required intensity of the bixels in light gray. Since the intensity of the dark gray bixel cannot be satisfied by those rectangles alone, it follows that at least one rectangle contained within the bounding box must be used to cover bixel $(6, 3)$. Let BB_{ij} represent the interior of a bounding box for bixel (i, j), i.e., given (l, u, r, d) all bixels at the intersection of rows $u + 1, \ldots, d - 1$ and columns $l + 1, \ldots, r - 1$. We denote the set of rectangles in $R(i, j)$ that are contained within BB_{ij} by $R(BB_{ij})$. In this case, the following inequality is valid:

$$\sum_{r \in R(BB_{ij})} y_r \geq 1. \tag{4–18}$$

Note that $(0, 0, n+1, m+1)$, which corresponds to the input matrix, is a bounding box for any bixel. Therefore there can be multiple bounding boxes associated with each bixel. Let BB_{ij} and BB'_{ij} be two bounding boxes for bixel (i, j). We say that BB_{ij} *dominates* BB'_{ij} if $R(BB_{ij}) \subset R(BB'_{ij})$. Since the inequality (4–18) that corresponds to a dominated

bounding box is implied by the inequality that is associated with the corresponding dominating bounding box, we are only interested in generating nondominated bounding boxes. Figure 4-5 displays another nondominated bounding box for the bixel considered in Figure 4-4.

2	3	**0**	8	2	4	2
2	1	**0**	5	1	2	1
3	**0**	**0**	5	**0**	**0**	3
5	**0**	2	8	6	**0**	3
0	8	14	10	9	**0**	3
5	8	20	7	1	**0**	4
5	9	5	4	**0**	**0**	3

Figure 4-5. Another nondominated bounding box seeded at (6,3)

To generate nondominated bounding boxes, we first make the following observation. A nondominated bounding box for bixel (i, j) is minimal in the sense that none of its edges can be shifted closer to (i, j) without violating the bounding box intensity property. We use this observation to design an algorithm that finds several nondominated bounding boxes associated with a given bixel. In our algorithm, we start at a bixel (i, j), and first move in a vertical or horizontal direction until we encounter a bixel (i', j') having $b_{i'j'} < b_{ij}$. We mark (i', j') as an edge of the bounding box, reduce b_{ij} by $b_{i'j'}$, and return to (i, j). We then move in the remaining directions one-by-one, updating b_{ij} after each step, to find the remaining edges of the bounding box. We repeat the same procedure for all 4! permutations of the directions, and obtain a nondominated bounding box in each iteration. Finally, we eliminate duplicates to obtain a set of nondominated bounding boxes, and we generate a constraint of type (4–18) for each bounding box.

4.2.2.3 Aggregate intensity inequalities

We derive a simple class of valid inequalities by observing that the total intensity that can be delivered to each bixel needs to be greater than or equal to its required intensity.

Formally,

$$\sum_{r \in R(i,j)} M_r y_r \geq b_{ij} \quad \forall i = 1, \ldots, m, \; j = 1, \ldots, n. \tag{4–19}$$

We note that inequalities (4–19) are implied by (4–2) and (4–3) in IPR. However, (4–19) can be used to tighten the master problem of the Benders decomposition approach discussed in Section 4.2.1. Furthermore, various tightening procedures can be applied to (4–19) for use in either the direct solution of IPR or in the Benders master problem. In our implementation, we apply a Chvátal-Gomory rounding procedure (see, e.g., Nemhauser and Wolsey (1988)) in which we divide both sides of the inequality by the smallest M_r coefficient on the left-hand-side (unless b_{ij} is divisible by that number), and round up coefficients on both sides of the inequality. If b_{ij} is divisible by the smallest M_r-coefficient on the left-hand-side of (4–19), then the rounding procedure yields an inequality implied by (4–19), and hence we do not generate it.

4.2.2.4 Special submatrices

An alternative strategy to the one described in Section 4.2.2.3 divides both sides of (4–19) by $b_{ij} - 1$, provided that $b_{ij} \geq 2$, and then rounds up all coefficients and the right-hand-side. Noting that all coefficients on the left-hand-side are bounded from above by b_{ij}, this process yields:

$$\sum_{\substack{r \in R(i,j): \\ M_r < b_{ij}}} y_r + 2 \sum_{\substack{r \in R(i,j): \\ M_r = b_{ij}}} y_r \geq 2 \quad \forall i = 1, \ldots, m, \; j = 1, \ldots, n. \tag{4–20}$$

Equations (4–20) imply that bixel (i, j) can either be covered by a single rectangle having a maximum intensity of b_{ij}, or otherwise needs to be covered by at least two rectangles. The idea behind (4–20) can be extended to other special cases. For instance, consider the following lemma.

Lemma 4. *Consider any* 1×2 *or* 2×1 *submatrix of* B *in which both elements equal a common nonzero value,* q*. Define* $A_1^=$ *as the set of rectangles that cover exactly one of the*

two bixels, and have a maximum intensity of q. Let $A_1^<$ be the set of all rectangles that cover exactly one of the two elements, and have a maximum intensity less than q. Define $A_2^=$ and $A_2^<$ analogously for rectangles that cover both elements. The following inequality is valid:

$$4\sum_{r\in A_2^=} y_r + 2\sum_{r\in A_2^<} y_r + 2\sum_{r\in A_1^=} y_r + \sum_{r\in A_1^<} y_r \geq 4. \tag{4–21}$$

Proof. Consider any feasible solution, and let vector $\mathbf{v}$ denote how many rectangles exist in the solution belonging to $A_2^=, A_2^<, A_1^=$, and $A_1^<$, respectively. We claim (without proof, for brevity) that the following vectors $\mathbf{v}_1, \ldots, \mathbf{v}_6$ are minimal, in the sense that $\mathbf{v} \geq \mathbf{v}_i$ for at least one $i = 1, \ldots, 6$, for every feasible $\mathbf{v}$: $\mathbf{v}_1 = (1,0,0,0), \mathbf{v}_2 = (0,1,0,2), \mathbf{v}_3 = (0,2,0,0), \mathbf{v}_4 = (0,0,1,2), \mathbf{v}_5 = (0,0,2,0), \mathbf{v}_6 = (0,0,0,4)$. Note that each solution represented by $\mathbf{v}_i$ satisfies (4–21), and thus all $\mathbf{v}$ corresponding to a feasible solution must also satisfy (4–21). □

Similarly, consider submatrices of the form

$$\begin{bmatrix} q_L & q_R \end{bmatrix},$$

or its transpose, where we assume $0 < q_L < q_R$ without loss of generality. We define $A_L^=$ and $A_L^<$ to be the sets of rectangles that cover q_L, but not q_R, with maximum intensity q_L, and less than q_L, respectively. Let $A_R^=$ and $A_R^<$ be defined for rectangles that cover q_R but not q_L, with a maximum intensity greater than or equal to $(q_R - q_L)$ and less than $(q_R - q_L)$, respectively. We define $A_2^=$ and $A_2^<$ as before, with a maximum intensity of q_L, and less than q_L, respectively. A similar analysis as in proof of Lemma 4 reveals that the following inequality is valid:

$$2\sum_{r\in A_L^=} y_r + 2\sum_{r\in A_R^=} y_r + 2\sum_{r\in A_2^=} y_r + \sum_{r\in A_L^<} y_r + \sum_{r\in A_R^<} y_r + \sum_{r\in A_2^<} y_r \geq 4. \tag{4–22}$$

The last special case that we consider is a nonzero submatrix of the form:

$$\begin{bmatrix} q & q \\ q & q \end{bmatrix}.$$

We define $A_i^{=}$ to be the sets of rectangles having maximum intensity equal to q, and covering exactly i elements of the 2×2 submatrix, for $i = 1$, 2, and 4. Similarly, define $A_i^{<}$ to be the sets of rectangles having maximum intensity less than q, and covering exactly i elements of the submatrix. Given these definitions, we obtain:

$$8 \sum_{r \in A_4^{=}} y_r + 4 \sum_{r \in A_4^{<}} y_r + 4 \sum_{r \in A_2^{=}} y_r + 2 \sum_{r \in A_2^{<}} y_r + 2 \sum_{r \in A_1^{=}} y_r + \sum_{r \in A_1^{<}} y_r \geq 8. \tag{4–23}$$

4.2.2.5 Submatrix inequalities

It is possible to generate valid inequalities using arguments similar to the ones discussed in Section 4.2.2.4 for other submatrices as well. However, this process is very tedious, and there is a large number of possible submatrix combinations. In this section we describe a similar set of inequalities, which are weaker than those described in the previous section, but are easier to generate. We first observe that the formulation IPR can be solved quickly for small input matrices. Let S denote a submatrix of the input matrix, and $R(S)$ represent the set of rectangles that cover at least one bixel in S. Let $LB(S)$ be a lower bound on the number of rectangles required to decompose S. Since $LB(S)$ constitutes a lower bound on the total number of rectangles required, the following inequality is valid for any submatrix S:

$$\sum_{r \in R(S)} y_r \geq \lceil LB(S) \rceil. \tag{4–24}$$

We can obtain $LB(S)$ by formulating an auxiliary integer programming problem of type IPR for S, and setting a limit on the maximum solution time.

4.2.3 Partitioning Approach

In this section, we propose a partitioning approach for our problem. We first propose an algorithm for detecting completely separable regions of the input matrix, which can be solved independently. Next, we explore methods for partitioning the large components, to obtain simultaneous upper and lower bounds, which we use to improve the solvability of our formulation.

4.2.3.1 Separable components

Our observations on clinical data sets suggest that input matrices can usually be decomposed into several small components, and one or two large components. The small components can usually be solved to optimality by formulation IPR enhanced with the valid inequalities discussed in Section 4.2.2.

We observe on clinical data that several regions of the input matrix are completely surrounded by zero-bixels. Since no rectangle can cover a zero-bixel, each of these regions can be solved independently. A *connected* subset of the input matrix obeys the property that a rectilinear path exists between any two nonzero-bixels of the subset, such that each bixel in the path is also a nonzero-bixel that belongs to the subset. We call a connected set of nonzero-bixels a *component* of the input matrix if it is adjacent to zero-bixels across all of its boundaries (i.e., if the subset is not contained within a larger connected subset).

To identify the components of the input matrix, we generate a graph G in which each nonzero-bixel has a corresponding node. We add an arc between a pair of nodes if and only if the corresponding bixels are adjacent in the input matrix. We then identify connected components on G by running a standard depth-first-search algorithm. Each connected component on G corresponds to a component of the input matrix, which can be solved independently of other components. Figure 4-6 depicts the components of the fluence map given in Figure 4-1.

Figure 4-6. Two components of a fluence map

4.2.3.2 Independent regions

After finding separable components of the input matrix, we attempt to further partition each component into smaller *regions*. We say that distinct regions of a component are *independent* if no rectangle intersects two bixels belonging to different regions without also intersecting a zero-bixel. In Figure 4-7, the regions with light and dark gray background are independent. If we solve IPR separately over all independent regions, the sum of rectangles required to decompose each independent region yields a lower bound on the objective function for the corresponding component.

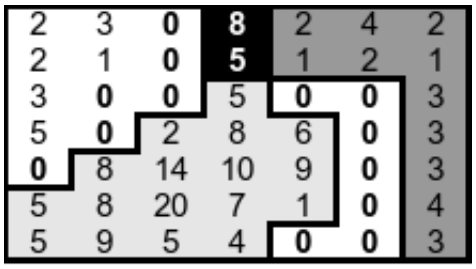


Figure 4-7. Regions of a connected component

In general, there are multiple ways of partitioning a component into independent regions, with each yielding possibly different lower bounds. The problem of finding a partition that yields the best lower bound can be thought of as a "dual" of finding the minimum number of rectangles to decompose a component. To solve this dual problem, we need to balance two conflicting criteria:

- The number of bixels assigned to each independent region needs to be small enough so that each region can be solved quickly.

- The number of bixels not assigned to any independent regions needs to be as small as possible to obtain a good lower bound.

We use a heuristic procedure to partition a component into independent regions, which employs an auxiliary objective of maximizing the number of component bixels covered by an independent region. Each bixel (i, j) is called "committed" if it either belongs to an independent region, or if (i, j) is contained within some rectangle in R that also covers bixels in an independent region (and hence, (i, j) cannot belong to another independent region). All other bixels are called "uncommitted." We select our independent regions one at a time, until no more uncommitted bixels remain. The procedure's details are described as follows.

Initialization. Labels all nonzero-bixels as "uncommitted."

Step 1. Each candidate independent region (or just "candidate") is seeded from a rectangle $r \in R$ such that rectangle r contains only uncommitted bixels, and such that the number of bixels in the rectangle is no more than some limit L. For each such rectangle r, define ℓ_r to be the (initial) candidate region.

Step 2. For each candidate ℓ_r, if ℓ_r covers exactly L bixels, then go to Step 4. Else, continue to Step 3.

Step 3. For each candidate ℓ_r, determine if there exists an uncommitted bixel (i, j) adjacent to ℓ_r (i.e., a bixel $(i, j) \notin \ell_r$ such that either $(i-1, j)$, $(i+1, j)$, $(i, j-1)$, or $(i, j+1)$ belongs to ℓ_r), such that for every $r' \in R(i, j)$, all bixels in r' either belong to ℓ_r, or would already become committed due to the selection of ℓ_r as an independent region. That is, adding (i, j) to ℓ_r would not increase the number of bixels committed by selecting ℓ_r as a new independent region. If such a bixel exists, then add (i, j) to ℓ_r, and return to Step 2. Else, continue to Step 4.

Step 4. For each candidate ℓ_r, compute κ_r^C = the number of bixels in ℓ_r, and κ_r^D = the number of uncommitted bixels (i, j) such that some rectangle in R includes both (i, j)

and a bixel in ℓ_r. If any candidates exist such that $\kappa_r^D = 0$, then choose $\ell_r^\star$ to be any such candidate. Else, choose $\ell_r^\star$ to be any candidate that maximizes κ_r^C/κ_r^D. Go to Step 5.

Step 5. Create an independent region corresponding to $\ell_r^\star$. For each bixel (i, j) that can be covered by a rectangle in R intersecting at least one bixel in $\ell_r^\star$, change the status of (i, j) to "committed." (This includes all bixels in $\ell_r^\star$ itself.) If all bixels are committed, terminate the procedure; else, return to Step 1.

In our algorithm for solving a component, we execute the foregoing heuristic to find a set of independent regions. We formulate IPR for each region, with a limit on the maximum solution time. We then use the lower bound obtained for each region to generate an inequality of type (4–24). (It is often prudent to skip this step if *only* one region is computed for a component.)

4.2.3.3 Dependent regions

In this section, we attempt to improve the lower bound obtained using independent regions by focusing on those bixels not included in the union of independent regions. We define a *dependent region* to be a connected set of bixels in a component that does not overlap with any of the independent regions in that component. In our example, the region with black background in Figure 4-7 is a dependent region. Let D represent the set of bixels in a dependent region, and let $\mathcal{R}(D)$ represent the set of rectangles that cover only a subset of the bixels in D.

To improve our lower bound, we wish to compute the minimum number of rectangles required to cover D; however, we wish to avoid double-counting those rectangles used to cover bixels in independent regions. Accordingly, we seek the minimum number of rectangles in $\mathcal{R}(D)$, perhaps in concert with rectangles outside $\mathcal{R}(D)$, required to cover the bixels in D. Using the x- and y-variables as before, we formulate the following variation of IPR to find the minimum number of rectangles in $\mathcal{R}(D)$ required to partition

D.

$$\textbf{DPR}: \text{Minimize} \quad \sum_{r \in \mathcal{R}(D)} y_r \tag{4–25}$$

$$\text{subject to:} \quad \sum_{r \in R(i,j)} x_r = b_{ij} \quad \forall (i,j) \in D \tag{4–26}$$

$$\sum_{r \in R(i,j)} x_r \leq b_{ij} \quad \forall i = 1, \ldots, m, \ j = 1, \ldots, n, \ (i,j) \notin D \tag{4–27}$$

$$x_r \leq M_r y_r \quad \forall r \in \mathcal{R}(D) \tag{4–28}$$

$$x_r \geq 0 \ \forall r \in R, \quad y_r \text{ binary } \forall r \in \mathcal{R}(D) \tag{4–29}$$

Objective (4–25) minimizes the number of rectangles in $\mathcal{R}(D)$ used in the solution. Constraints (4–26) ensure that the bixels in D get partitioned exactly, where (4–27) limit the intensity delivered to the remaining bixels. Constraints (4–28) relate the x- and y-variables as done in IPR, and finally (4–29) define variable types. As before, we set a time limit for the solution of DPR, and obtain a lower bound on the objective function value, which we denote by $LB(D)$. Given this value, the following inequality is valid:

$$\sum_{r \in \mathcal{R}(D)} y_r \geq \lceil LB(D) \rceil. \tag{4–30}$$

In our example, the optimal value of DPR for the black (dependent) region is 1 since the intensity requirement of bixel $(1,4)$ cannot be satisfied completely by rectangles that cover bixels in the gray (independent) regions (in fact, this result can also be seen due to the bounding box constraint implying that one rectangle representing the singleton bixel (1,4) must appear in any feasible solution). We note that the rectangles in $\mathcal{R}(D)$, by definition, do not intersect any other (dependent or independent) regions. Therefore, the lower bounds obtained for all regions can be summed to obtain a lower bound on the minimum number of rectangles required to decompose a component.

4.2.3.4 Upper bound calculation

In this section, we discuss how a related approach leads to a heuristic algorithm to obtain a feasible decomposition of a component. We first note that a feasible decomposition of a component can be obtained by combining feasible solutions obtained for individual regions within a component. Feasible solutions for independent regions are readily available from the integer programming problems solved for obtaining lower bounds on those regions, as discussed in Section 4.2.3.2. Feasible solutions for dependent regions can be extracted from solutions of the formulation given by DPR. However, since DPR minimizes the number of rectangles that are contained within a dependent region, and not necessarily the total number of rectangles required to decompose a dependent region, the solutions obtained from DPR potentially use an unnecessarily large number of rectangles not contained in $\mathcal{R}(D)$.

A better way of obtaining feasible solutions for dependent regions is to formulate the problem IPR for each dependent region. Since IPR explicitly minimizes the total number of rectangles required, we expect this approach to result in feasible solutions of higher quality. However, this approach does not consider the fact that some of the rectangles that are already used for decomposing independent regions can be extended into dependent regions without increasing the total number of rectangles. To permit the use of rectangles that intersect independent and dependent regions, we require a revised integer programming formulation.

In our approach, we solve the integer programming formulations for decomposing the independent regions first, and store the best feasible solutions found within the allowed time limit. Let $\bar{x}_r$ represent the intensity assigned to rectangle r for decomposing independent regions. Next, we generate a feasible solution for each dependent region, one at a time, as follows. We first find the set of rectangles that can be extended into the current dependent region, and determine how those rectangles can be extended. Let $E(D, r)$ represent the set of rectangles in R that extend rectangle r into dependent region

D. We also define the parameter $I(r)_{ij}^{e}$ equal to one if bixel $(i,j) \in D$ is covered by extension e of rectangle r, and zero otherwise. Let z_{re} be a binary variable that equals 1 if and only if extension e of rectangle r is used in the solution. We define the x- and y-variables as before, and formulate the following problem:

$$\textbf{EPR}\text{: Minimize} \quad \sum_{r \in \mathcal{R}(D)} y_r \tag{4–31}$$

$$\text{subject to:} \quad \sum_{r \in R(i,j)} x_r = b_{ij} - \sum_{r \in R} \sum_{e \in E(D,r)} \left(\bar{x}_r I(r)_{ij}^{e}\right) z_{re} \quad \forall (i,j) \in D \tag{4–32}$$

$$\sum_{e \in E(D,r)} z_{re} \leq 1 \quad \forall r \in R \tag{4–33}$$

$$x_r \leq M_r y_r \quad \forall r \in \mathcal{R}(D) \tag{4–34}$$

$$x_r \geq 0,\ y_r \text{ binary} \quad \forall r \in \mathcal{R}(D) \tag{4–35}$$

$$z_{re} \text{ binary} \quad \forall r \in R,\ e \in E(D,r). \tag{4–36}$$

We generate a feasible solution by combining three types of rectangles: (i) rectangles used to decompose independent regions that are not extended by EPR; (ii) rectangles obtained by extending rectangles from independent regions into dependent regions by EPR; and (iii) rectangles in $\mathcal{R}(D)$ used by EPR.

Note that the optimal value of EPR for the dependent region given in Figure 4-7 is 1. This can be seen by observing that the rectangle(s) that cover bixel $(3,4)$ can be extended up to fully satisfy the intensity requirement of bixel $(2,4)$ without any penalty on the objective function of EPR formulated for the dependent region. Therefore, a single rectangle contained in the dependent region solves EPR optimally. Since the optimal value of DPR for the dependent region is also 1, our partition solves the problem of finding the minimum number of rectangles to optimality.

4.3 Extensions

In this section, we briefly discuss how our model can be adjusted to tackle the problems of minimizing total treatment time, and lexicographically minimizing beam-on-time and number of apertures.

4.3.1 Minimize Total Treatment Time

The total time spent delivering a given fluence map is composed of (i) time required to move the jaws to form the next rectangular aperture (setup time), and (ii) time during which radiation is delivered (beam-on-time). Even though the setup time required for switching from one rectangular aperture to the next one depends on the jaw settings corresponding to these apertures, and hence is sequence-dependent, we make the common assumption that total setup time is proportional to the total number of apertures used. With this assumption, our model can easily be adjusted to explicitly minimize the total treatment time by changing the objective function of IPR to

$$\text{Minimize } w \sum_{r \in R} y_r + \sum_{r \in R} x_r, \tag{4–37}$$

where w is a parameter that represents the average setup time per aperture relative to the time required to deliver a unit of intensity.

The Benders decomposition procedure discussed in Section 4.2.1 also needs to be adjusted accordingly. We first add a continuous variable t to MP, which "predicts" the minimum beam-on-time that can be obtained by the set of rectangles chosen by MP. The updated master problem can be written as follows.

$$\textbf{MPTT: Minimize } w \sum_{r \in R} y_r + t \tag{4–38}$$

$$\text{subject to: } \mathbf{y} \text{ corresponds to a feasible decomposition} \tag{4–39}$$

$$t \geq \text{minimum beam-on-time corresponding to } \mathbf{y} \tag{4–40}$$

$$y_r \text{ binary} \quad \forall r \in R. \tag{4–41}$$

Given a vector $\hat{\mathbf{y}}$, we can find the minimum beam-on-time for the corresponding decomposition, if one exists, by solving:

$$\textbf{SPTT}(\hat{\mathbf{y}})\text{: Minimize } \sum_{r\in R} x_r \tag{4–42}$$

$$\text{subject to: } \sum_{r\in R(i,j)} x_r = b_{ij} \quad \forall i = 1,\ldots,m,\ j = 1,\ldots,n \tag{4–43}$$

$$x_r \leq M_r \hat{y}_r \quad \forall r \in R \tag{4–44}$$

$$x_r \geq 0 \quad \forall r \in R. \tag{4–45}$$

Note that SPTT is obtained by simply changing the objective function of SP. If SPTT($\hat{\mathbf{y}}$) is infeasible, then we add a Benders feasibility cut of type (4–16) as before, and re-solve MPTT. Otherwise, let the value of t in MPTT be $\hat{t}$, and the optimal objective function value of SPTT be $t^\star$. If $\hat{t} = t^\star$, then $(\hat{\mathbf{y}}, \hat{t})$ is an optimal solution of MPTT that minimizes the total treatment time. However, if $\hat{t} > t^\star$, then we add the following Benders optimality cut

$$t \geq \sum_{i=1}^{m}\sum_{j=1}^{n} b_{ij}\hat{\alpha}_{ij} + \sum_{r\in R}(M_r\hat{\beta}_r)y_r, \tag{4–46}$$

where $\hat{\alpha}_{ij}$ and $\hat{\beta}_r$ are optimal dual multipliers associated with constraints (4–43) and (4–44), respectively.

4.3.2 Optimization with Beam-on-Time Restrictions

Another related problem that we consider is finding the minimum number of rectangles that yields the minimum beam-on-time. Note that the minimum beam-on-time required to decompose a fluence map can be found (in polynomial time) by solving SPTT, which is a linear program, for the vector $\hat{y}_r = 1,\ \forall r \in R$. Let $T^\star$ denote the optimal objective function value of SPTT($\vec{1}$), where $\vec{1}$ is the vector of all 1's. Given this value, it is sufficient to add

$$\sum_{r\in R} x_r \leq T^\star \tag{4–47}$$

to minimize the number of rectangles while limiting beam-on-time to $T^\star$.

The modifications required for the Benders decomposition algorithm are also straightforward. To enforce the minimum beam-on-time restriction, we add (4–47) to SP, which checks whether a given set of rectangles can decompose the fluence map. The updated feasibility cut is given by

$$\sum_{i=1}^{m}\sum_{j=1}^{n} b_{ij}\hat{\alpha}_{ij} + \sum_{r\in R}(M_r\hat{\beta}_r)y_r + T^\star\hat{\theta} \leq 0, \tag{4–48}$$

where θ is the dual variable associated with (4–47) in SP. Finally, we need to check whether the solution generated by our heuristic discussed in Section 4.2.3.4 satisfies constraint (4–47); if so, then it can be used as an initial upper bound.

4.4 Computational Results

We have implemented our algorithms using CPLEX 11 running on a Windows XP PC with a 3.4 GHz CPU and 2 GB RAM. Our base set of test problem instances consists of 25 clinical problem instances ("c1b1", ..., "c5b5"). These instances were obtained from treatment plans for five patients treated using five beam angles each. We report problem characteristics in terms of the number of rows m, the number of columns n, and the maximum intensity value L. We imposed a time limit of 1800 seconds (30 minutes) in all of our tests. For problem instances that were not solved to optimality within the imposed time limit, we report the best upper and lower bounds obtained, where we round lower bounds up for the cases in which the objective function is guaranteed to have an integral value.

Our preliminary computational tests showed that the naive implementation of our Benders decomposition approach, in which we add a cut and re-solve the master problem in each iteration, was not computationally competitive with solving the explicit integer programming formulation. This is due to the fact that repetitively solving the master problem, which is an integer programming problem, is computationally very expensive. We instead used callback functions of CPLEX to generate a single branch-and-bound

tree in which we solve SP($\hat{\mathbf{y}}$) (or (SPTT($\hat{\mathbf{y}}$)) corresponding to each integer solution found in the branch-and-bound tree, and add cuts to tighten the master problem as necessary. While this approach produced better results than the naive implementation, it still yielded inferior bounds than those obtained from the explicit formulation. Therefore, we omit further Benders-based computational results.

Our first experiment quantifies the effects of the valid inequalities discussed in Section 4.2.2, and the partitioning approach discussed in Section 4.2.3 on solution quality and execution time. In Table 4-1, the set of columns labeled "Default CPLEX" shows the results we obtained by solving the formulation IPR on each problem instance using default CPLEX options. The "+ Valid inequalities" columns represent the IPR formulation enhanced with the adjacent rectangle inequalities (4–17), bounding box inequalities (4–18), strengthened aggregate intensity inequalities (4–19) and (4–20), and 1×2 submatrix inequalities (4–21) and (4–22). (Additional computational results showed that the 2×2 submatrix inequalities (4–23) and the arbitrary submatrix inequalities (4–24) did not improve the solvability of the model.) The set of columns labeled "+ Partitions" shows the results we obtained by partitioning the problem into separable components (Section 4.2.3.1), further partitioning each component into independent and dependent regions (Sections 4.2.3.2 and 4.2.3.3), and using our upper bounding heuristic (Section 4.2.3.4) in addition to the valid inequalities used for the tests in the previous set of columns. We refer to the latter settings as our *base algorithm* in the remaining computational tests.

Each set of columns in Table 4-1 displays the time spent for each problem instance ("CPU"), and upper bound ("UB"), lower bound ("LB"), and optimality gap ("GAP") obtained. We also report the average and maximum gaps over all problem instances. We observe that none of the problem instances were solved to optimality using the default CPLEX options, whereas c1b2 and c5b2 were solved to optimality after adding the valid inequalities of Section 4.2.2. An additional instance (c5b5) was solved using the partitioning strategy described in Section 4.2.3. We note that even though our approach

Table 4-1. Effect of valid inequalities and the partitioning strategy

				Default CPLEX				+ Valid inequalities				+ Partitions			
Name	m	n	L	CPU	UB	LB	Gap	CPU	UB	LB	Gap	CPU	UB	LB	Gap
c1b1	15	14	20	1800	66	60	0.09	1800	63	62	**0.02**	1800	64	62	0.03
c1b2	11	15	20	1800	48	47	0.02	138.2	48	48	**0**	1009.9	48	48	**0**
c1b3	15	15	20	1800	57	54	0.05	1800	57	54	**0.05**	1800	58	54	0.07
c1b4	15	15	20	1800	61	52	0.15	1800	61	53	0.13	1800	59	55	**0.07**
c1b5	11	15	20	1800	47	45	0.04	1800	46	45	**0.02**	1800	47	45	0.04
c2b1	18	20	20	1800	114	79	0.31	1800	119	85	0.29	1800	103	87	**0.16**
c2b2	17	19	20	1800	95	69	0.27	1800	96	81	0.16	1800	94	82	**0.13**
c2b3	18	18	20	1800	98	73	0.26	1800	103	77	0.25	1800	94	77	**0.18**
c2b4	18	18	20	1800	114	80	0.3	1800	115	84	0.27	1800	105	88	**0.16**
c2b5	17	18	20	1800	94	64	0.32	1800	98	72	0.27	1800	91	72	**0.21**
c3b1	22	17	20	1800	121	69	0.43	1800	134	79	0.41	1800	119	79	**0.34**
c3b2	15	19	20	1800	73	46	0.37	1800	71	52	0.27	1800	70	52	**0.26**
c3b3	20	17	20	1800	119	69	0.42	1800	119	75	0.37	1800	107	77	**0.28**
c3b4	19	17	20	1800	103	69	0.33	1800	106	73	0.31	1800	99	78	**0.21**
c3b5	15	19	20	1800	73	55	0.25	1800	71	58	**0.18**	1800	73	58	0.21
c4b1	19	22	20	1800	106	79	0.25	1800	107	89	**0.17**	1800	109	89	0.18
c4b2	13	24	20	1800	88	54	0.39	1800	99	58	0.41	1800	91	58	**0.36**
c4b3	18	23	20	1800	95	71	0.25	1800	99	75	0.24	1800	93	77	**0.17**
c4b4	17	23	20	1800	103	78	0.24	1800	102	81	0.21	1800	98	83	**0.15**
c4b5	18	24	20	1800	93	62	0.33	1800	93	66	0.29	1800	87	67	**0.23**
c5b1	15	16	20	1800	66	64	0.03	1800	66	65	**0.02**	1800	66	65	**0.02**
c5b2	13	17	20	1800	58	57	0.02	102.1	58	58	**0**	213.6	58	58	**0**
c5b3	14	16	20	1800	63	54	0.14	1800	68	56	0.18	1800	65	57	**0.12**
c5b4	14	16	20	1800	63	57	0.1	1800	64	59	0.08	1800	62	59	**0.05**
c5b5	12	17	20	1800	53	47	0.11	1800	51	48	0.06	36.2	49	49	**0**

was not able to provide provably optimal solutions for most instances, it significantly improved both lower and upper bounds for several instances.

Our next experiment tests our base algorithm under the extensions discussed in Section 4.3. The set of columns labeled as "Total Time" in Table 4-2 presents the extension in which the objective function is defined as a linear combination of the beam-on-time and the number of rectangles. The actual value of w depends on the particular treatment delivery equipment used in the clinic, where values of w in the range 1–10 are typical (see, e.g., Dai and Hu (1999), and Taşkın et al. (2009b)). In our experiments, we therefore used $w = 7$ as a representative value. The next set of columns ("Lexicographic") is dedicated to the extension in which we first minimize beam-on-time, $T^\star$, and then find the minimum number of rectangles that yields the minimum beam-on-time. The column "BOT" represents the value of $T^\star$, and "Total Time" represents the total treatment time associated with the solution found, where we again use $w = 7$ as the average setup time per rectangle. We observe that our algorithm could solve more problem instances to optimality for both extensions compared to the

Table 4-2. Computational results on model extensions

				Total Time				Lexicographic					
Name	m	n	L	CPU	UB	LB	Gap	CPU	UB	LB	Gap	BOT	Total Time
c1b1	15	14	20	255.9	621	621	0	36.5	66	66	0	176	638
c1b2	11	15	20	330.3	459	459	0	132.6	50	50	0	121	471
c1b3	15	15	20	1800	548	542.72	0.01	130.4	62	62	0	147	581
c1b4	15	15	20	1800	557	542.49	0.03	186.9	62	62	0	136	570
c1b5	11	15	20	1800	451	443.63	0.02	30.9	53	53	0	115	486
c2b1	18	20	20	1800	962	814.24	0.15	1800	107	104	0.03	194	943
c2b2	17	19	20	1800	883	797.74	0.1	1800	96	92	0.04	207	879
c2b3	18	18	20	1800	918	797.6	0.13	1800	96	88	0.08	237	909
c2b4	18	18	20	1800	1028	889.36	0.13	1800	111	106	0.05	258	1035
c2b5	17	18	20	1800	890	721.13	0.19	1800	92	83	0.1	207	851
c3b1	22	17	20	1800	1161	858.9	0.26	1800	116	103	0.11	266	1078
c3b2	15	19	20	1800	668	533.24	0.2	1800	70	64	0.09	151	641
c3b3	20	17	20	1800	1066	847.09	0.21	1800	111	95	0.14	278	1055
c3b4	19	17	20	1800	1023	857.91	0.16	1800	103	95	0.08	287	1008
c3b5	15	19	20	1800	722	610.01	0.16	204.4	76	76	0	182	714
c4b1	19	22	20	1800	1044	918.57	0.12	1800	108	105	0.03	275	1031
c4b2	13	24	20	1800	895	656.15	0.27	1800	95	76	0.2	232	897
c4b3	18	23	20	1800	858	743.62	0.13	1800	92	89	0.03	189	833
c4b4	17	23	20	1800	943	834.32	0.12	1800	101	96	0.05	235	942
c4b5	18	24	20	1800	913	740.19	0.19	1800	86	77	0.1	260	862
c5b1	15	16	20	271.4	626	626	0	5.5	71	71	0	158	655
c5b2	13	17	20	33.4	597	597	0	19.9	63	63	0	156	597
c5b3	14	16	20	1800	623	597.96	0.04	869.2	68	68	0	180	656
c5b4	14	16	20	1800	584	571.15	0.02	192.4	66	66	0	145	607
c5b5	12	17	20	90.4	503	503	0	37.2	57	57	0	147	546

problem of finding the minimum number of rectangles. To understand why this is the case, we first note that the difficulty of the matrix decomposition problem varies greatly based on the objective function used. On one hand, minimizing the number of rectangles is strongly NP-hard, even for fluence maps having a single row (see Baatar et al. (2005)). On the other hand, minimizing the beam-on-time is a polynomially solvable problem (see Section 4.3). Therefore, we expect that the problem should become easier as the weight of the beam-on-time term in the objective function increases. The reason the lexicographic minimization problem is easier to solve than the other two variations is because the additional beam-on-time constraint considerably shrinks the feasible solution space.

Another way of looking at the problem of balancing the number of apertures and the beam-on-time is to view the problem as a multicriteria optimization problem. In this setting, we are interested in constructing the Pareto efficient frontier of solutions with the property that neither of the two criteria can be improved without deteriorating the other. Note that the lexicographic approach that we considered above determines a particular Pareto optimal solution to the multicriteria problem. To generate other non-dominated

solutions for the multicriteria version of the problem, we sequentially impose different upper bounds on the number of apertures allowed, say γ, and find the corresponding minimum beam-on-time for these values of γ. As an example, we considered the problem instance c5b5. For this instance, we note that the minimum number of apertures is 49 (see Table 4-1) with a corresponding beam-on-time of 160, while the minimum beam-on-time for this problem instance is 147 (see Table 4-2) which requires 57 apertures. Figure 4-8 then depicts (i) the non-dominated solutions; (ii) the Pareto efficient frontier for values of $\gamma \in [49, 57]$, and (iii) the (boundary of the) convex hull of the Pareto set. The solutions on the latter are the optimal solutions to the problem of minimizing total treatment time that can be obtained with different values of w.

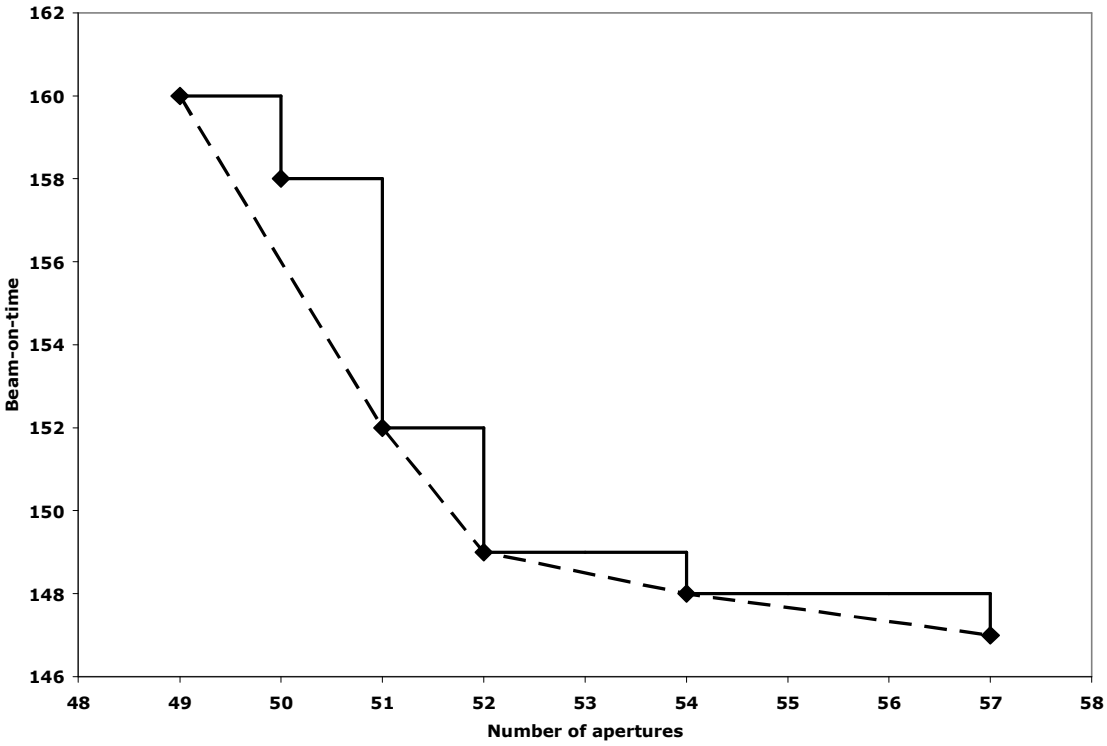


Figure 4-8. Efficient frontier for number of apertures and beam-on-time

Our final experiment analyzes the effect of the maximum intensity value L. Usually fluence maps are obtained by solving a nonlinear optimization problem for each beam angle to determine an intensity profile for each beam angle, which is represented by a

Table 4-3. Effect of maximum intensity value on solvability

			$L = 5$				$L = 10$				$L = 15$			
Name	m	n	CPU	UB	LB	Gap	CPU	UB	LB	Gap	CPU	UB	LB	Gap
c1b1	15	14	4.4	305	305	0	22.9	441	441	0	1800	539	536.99	0
c1b2	11	15	1.4	238	238	0	4.4	320	320	0	498.5	394	394	0
c1b3	15	15	9.6	287	287	0	228.6	377	377	0	1800	495	487.67	0.01
c1b4	15	15	5.8	269	269	0	1800	393	377.42	0.04	1800	513	493.13	0.04
c1b5	11	15	2.4	216	216	0	28.9	326	326	0	316.7	411	411	0
c2b1	18	20	31.3	440	440	0	1800	648	635.1	0.02	1800	826	732.68	0.11
c2b2	17	19	51.5	448	448	0	1800	625	599.84	0.04	1800	759	690.67	0.09
c2b3	18	18	144.4	428	428	0	1800	645	615.64	0.05	1800	800	704.61	0.12
c2b4	18	18	1593.1	487	487	0	1800	755	678.49	0.1	1800	919	796.13	0.13
c2b5	17	18	197.9	429	429	0	1800	606	538.62	0.11	1800	728	621.9	0.15
c3b1	22	17	1359.4	480	480	0	1800	747	662.47	0.11	1800	1080	779.16	0.28
c3b2	15	19	1800	280	274.97	0.02	1800	414	376.01	0.09	1800	532	461.6	0.13
c3b3	20	17	1800	461	446.77	0.03	1800	731	641.2	0.12	1800	908	755.25	0.17
c3b4	19	17	1800	463	456.42	0.01	1800	713	634.26	0.11	1800	900	762.02	0.15
c3b5	15	19	1800	332	325.87	0.02	1800	481	466.6	0.03	1800	582	532.2	0.09
c4b1	19	22	39.9	529	529	0	1800	758	719.53	0.05	1800	899	827.73	0.08
c4b2	13	24	1800	422	408.14	0.03	1800	595	503.92	0.15	1800	764	582.69	0.24
c4b3	18	23	126.3	409	409	0	1800	579	564.5	0.03	1800	695	666.23	0.04
c4b4	17	23	321.4	444	444	0	1800	662	649.1	0.02	1800	815	742.81	0.09
c4b5	18	24	1194.7	414	414	0	1800	636	573.02	0.1	1800	794	662.74	0.17
c5b1	15	16	5.9	342	342	0	5.6	442	442	0	28.9	584	584	0
c5b2	13	17	4.3	289	289	0	5.5	439	439	0	49.3	516	516	0
c5b3	14	16	1574.4	294	294	0	1800	453	441.44	0.03	1800	566	538.22	0.05
c5b4	14	16	3.6	239	239	0	1800	473	468.49	0.01	1800	534	524.31	0.02
c5b5	12	17	2.1	252	252	0	9.3	354	354	0	63.9	441	441	0

matrix of real numbers. Later, these matrices are rounded to integer matrices to limit the delivery time. To analyze the trade-off between round-off errors and treatment time, we started from clinical treatment plans, and applied rounding with different levels of granularity. Specifically, we generated problem instances from the same fluence maps with $L \in \{5, 10, 15, 20\}$, and used our algorithm to find the minimum total treatment time required as a measure of delivery efficiency. Table 4-3 shows the results of our experiments. We observe that our algorithm produces smaller optimality gaps as L decreases, which is not surprising since IPR becomes tighter as the M_r-coefficients (which are bounded by L) decrease. Furthermore, delivery efficiency is also higher for small values of L. The average treatment time (calculated over the lower bounds) for all problem instances increases from 366.05 for $L = 5$ to 513.79 for $L = 10$, 609.79 for $L = 15$, and 684.96 for $L = 20$, which is calculated using the set of columns labeled "Total Time" in Table 4-2. Our results show that the choice of granularity chosen for rounding has a significant effect on the treatment time. For each individual patient, the risks associated with the deterioration in treatment plan quality due to the rounding of intensities needs

to be weighed against the disadvantages of a longer treatment time by the physician or clinician.

CHAPTER 5
GRAPH SEARCH PROBLEM

5.1 Introduction and Literature Review

In this chapter we consider several variants of a search problem on graphs, which can be seen as a game between an *intruder* and a group of *searchers*. Consider an undirected graph $G = (N, E)$. The intruder and searchers occupy some nodes of the graph. At each time period, a player (intruder or searcher) located at node $i \in N$ can move along an edge to an adjacent node, or stay at their current position. A searcher located at node $i \in N$ can detect an intruder if it is located at some node in $S(i) \subseteq N$. Searchers can be deployed at any node, but have no information about the location of the intruder. Therefore, they have to systematically search the graph to be able to detect the intruder, even if the intruder has perfect information about the searchers' plan, and utilizes this knowledge to evade detection for as long as possible. The problem is to find the minimum number of searchers needed and a routing plan for each searcher that guarantees detection of the intruder within a given time limit.

The graph search problem was initially defined by Parsons (1978) in the context of seeking a person lost in a cave. The cave is represented as a graph, where tunnels of the cave correspond to edges of the graph. Searchers have to sweep edges of the graph to locate the missing person, who is assumed to be wandering unpredictably or is purposefully trying to evade searchers. The *search number* $s(G)$ of a graph G is defined to be the minimum number of searchers needed so that the missing person can be found even if he could move infinitely fast along any path not occupied by searchers (Parsons, 1978). Computing $s(G)$ is NP-hard for general graphs (Bienstock and Seymour, 1991; LaPaugh, 1993; Megiddo et al., 1988), but it can be computed in linear time for trees (Alspach, 2004; Megiddo et al., 1988; Peng et al., 2000). The search number of a graph has been shown to be related to other important parameters such as tree-width, path-width, and vertex separation (Dendris et al., 1997; Ellis et al., 1994; Seymour and Thomas, 1993).

Several variants of the graph search problem have been investigated in the literature. In decontamination problems, edges or nodes of a graph are infected by a contaminant such as a computer virus or a chemical agent, which spreads across the graph (Flocchini et al., 2008; LaPaugh, 1993; Penuel and Smith, 2009). In rendezvous problems different players, who are not aware of the location of others, try to meet at a common node as quickly as possible (Alpern, 1995; Alpern and Gal, 2003; Kikuta and Ruckle, 2007). Hide-and-seek problems consider an intruder that "hides" in a stationary location, while the searchers try to locate the intruder in minimum time (Alpern, 2008; Jotshi and Batta, 2008). Such problems also arise in search-and-rescue settings (Benkoski et al., 1991). Pursuit evasion (or "cops-and-robber") games model an intruder that tries to avoid being captured by searchers (Aigner and Fromme, 1984; Alspach et al., 2008; Hahn, 2007; Isler and Karnad, 2008). In some applications nodes of a graph need to be patrolled for protection or supervision (Chevaleyre et al., 2004; Sak et al., 2008). In particular, one interesting application coordinates automated software searchers so that they patrol the Internet to find web sites that exploit browser vulnerabilities (Wang et al., 2005). We refer the reader to Alpern and Gal (2003); Alspach (2004); Fomin and Thilikos (2008) for detailed surveys of the literature on search problems and applications in various practical settings.

Most of the previous research on graph search problems has focused on theoretical aspects of the problems (e.g. Chevaleyre et al. (2004); Dendris et al. (1997); Ellis et al. (1994); Goldstein and Reingold (1995); Seymour and Thomas (1993)) or designing algorithms for solving the problems on special graph structures (e.g. Alpern (2008); Flocchini et al. (2008); Kikuta and Ruckle (2007); Peng et al. (2000)). Our contribution is an exact optimization algorithm for solving several variants of the search problem on general graphs (see also Penuel and Smith (2009) for a decontamination problem in which the intruder location has been determined). In particular, we consider three specific graph search problems: (i) a hide-and-seek problem, (ii) a pursuit evasion problem, and (iii) a

patrol problem. We model these problems as large-scale integer programs, and propose a branch-cut-price algorithm that is capable of solving all three problems with some modifications.

A variant of the branch-and-bound algorithm, which adds cutting planes to linear programming relaxations to tighten dual bounds is called branch-and-cut, and is employed in most commercial solvers for solving integer programs (Marchand et al., 2002; Nemhauser and Wolsey, 1988; Wolsey, 1998). An effective method for solving integer programs having a large number of variables is branch-and-price, which is based on dynamic column generation (Barnhart et al., 1998). Branch-cut-price is essentially an algorithm that combines dynamic column generation with dynamic row generation (Jünger and Thienel, 2000).

The remainder of this chapter is organized as follows. In Section 5.2 we describe a hide-and-seek problem and propose a column generation algorithm for solving its linear programming relaxation. Similarly, Sections 5.3 and 5.4 analyze the pursuit evasion and patrol problems, respectively. We describe some branching rules that can be used in all three algorithms to obtain an optimal solution to these problems in Section 5.5. Finally, we give computational results in Section 5.6.

5.2 Hide-and-Seek Problem

We start our discussion with a hide-and-seek problem, in which a group of searchers seek to locate a stationary intruder within T time steps. While this problem is relatively easy to model and analyze, it serves as the basis for our more complex search algorithms.

5.2.1 Mathematical Model

We denote the set of nodes adjacent to node $i \in N$ by $A(i)$. Since we allow the intruder and searchers to stay at their current location, we assume that $i \in A(i),\ \forall i \in N$. Recall that a walk on a graph $G = (N, E)$ is a sequence $i_1, \ldots, i_r$ of nodes such that for all $1 \leq k \leq r-1$, $i_{k+1} \in A(i_k)$ (Ahuja et al., 1993). We define the *length* of a walk as the number of traversed edges on the walk. Let $P(T)$ denote the set of all possible walks of

length at most T that can be taken by a searcher. We assume that each node $i \in N$ can be observed from some node, i.e., there exists a $j \in N$ such that $j \in S(i)$. Let d_{pi} be a parameter whose value is 1 if a searcher following walk p can detect an intruder located at node i, and 0 otherwise. Let λ_p be a binary variable that equals 1 if a searcher is assigned to follow walk p, and 0 otherwise. Given these definitions, our hide-and-seek problem can be formulated as the following set covering problem.

$$\textbf{HS: minimize} \quad \sum_{p \in P(T)} \lambda_p \tag{5–1}$$

$$\text{subject to} \quad \sum_{p \in P(T)} d_{pi} \lambda_p \geq 1 \quad \forall i \in N \tag{5–2}$$

$$\lambda_p \in \{0, 1\} \quad \forall p \in P(T) \tag{5–3}$$

The objective function (5–1) minimizes the number of selected searchers, and constraints (5–2) guarantee that each node is covered by at least one searcher within the allowed time frame. We note that a special case of this problem for which a searcher located at node $i \in N$ can observe node i and its neighbors, and we need to guarantee immediate detection of the intruder (i.e., when $S(i) = A(i),\ \forall i \in N$ and $T = 0$), is equivalent to the minimum dominating set problem, which is known to be NP-hard (Garey and Johnson, 1979). Therefore, the hide-and-seek problem that we consider is NP-hard.

5.2.2 Solution Approach

In principle, all walks of length at most T can be enumerated, and HS can be solved directly. This approach might be practical for small values of T, but in general there is an exponential number of such walks, which corresponds to an exponential number of variables. We instead propose a column generation approach to solve HS. Given a subset of walks $P'(T) \subseteq P(T)$, we can construct a limited hide-and-seek (LHS) problem identical to HS, with $P(T)$ replaced by $P'(T)$. The linear programming relaxation of LHS is:

$$\textbf{LHSLP: minimize} \quad \sum_{p \in P'(T)} \lambda_p \tag{5–4}$$

$$\text{subject to} \sum_{p \in P'(T)} d_{pi}\lambda_p \geq 1 \quad \forall i \in N \tag{5–5}$$

$$\lambda_p \geq 0 \quad \forall p \in P'(T), \tag{5–6}$$

where upper bounds on the λ-variables are not necessary at optimality. Given an optimal dual vector $\hat{\gamma}$, the reduced cost of λ_p, which we denote by $\bar{c}_p$, can be calculated as $1 - \sum_{i \in N} \hat{\gamma}_i d_{pi}$. Since $\hat{\gamma}$ is an optimal dual vector, $\bar{c}_p \geq 0$ for all $p \in P'(T)$. We can conclude that the current solution of LHSLP is also optimal for the linear programming relaxation of HS if $\bar{c}_p \geq 0$ for all $p \in P(T)$. On the other hand, if $\bar{c}_{\hat{p}} < 0$ for some $\hat{p} \in P(T) \setminus P'(T)$, then adding $\hat{p}$ to $P'(T)$ can potentially decrease the value of the objective function (5–4). We discuss our pricing problem, which seeks such a $\hat{p}$, in the next section.

5.2.2.1 Searcher's problem

Let γ_i be the dual variable associated with the constraint of type (5–5) corresponding to node $i \in N$. Also, let y_i be a decision variable that equals 1 if node i is "seen" by a searcher following a walk that we generate, and 0 otherwise, $\forall i \in N$. Given an optimal dual vector $\hat{\gamma}$, we solve the following pricing problem to seek a λ-variable having a negative reduced cost: max $\sum_{i \in N} \hat{\gamma}_i y_i$, subject to the restriction that $(y_1, \ldots, y_{|N|})$ corresponds to a set of nodes observed by a walk of length no more than T.

The pricing problem can be formulated as a mixed-integer programming problem on a time-expanded network consisting of $T + 1$ stages. In particular, we create a node N_{it} for each $i \in N,\ t = 0, \ldots, T$. We create an arc from node N_{it}, $\forall i \in N,\ t = 0, \ldots, T - 1$ to nodes $N_{j(t+1)}$ for all $j \in A(i)$. For this problem it is easy to see that an optimal solution exists in which all searchers move at each time period. Therefore, we omit arcs between nodes N_{it} and $N_{i(t+1)}$ for each $i \in N$. Figure 5-1 displays a simple example graph, and the corresponding time-expanded network for $T = 2$.

To formulate the pricing problem as a mixed-integer program, we introduce binary variables $x_i^t = 1$ if the searcher is at node i at time t. Then, an integer programming

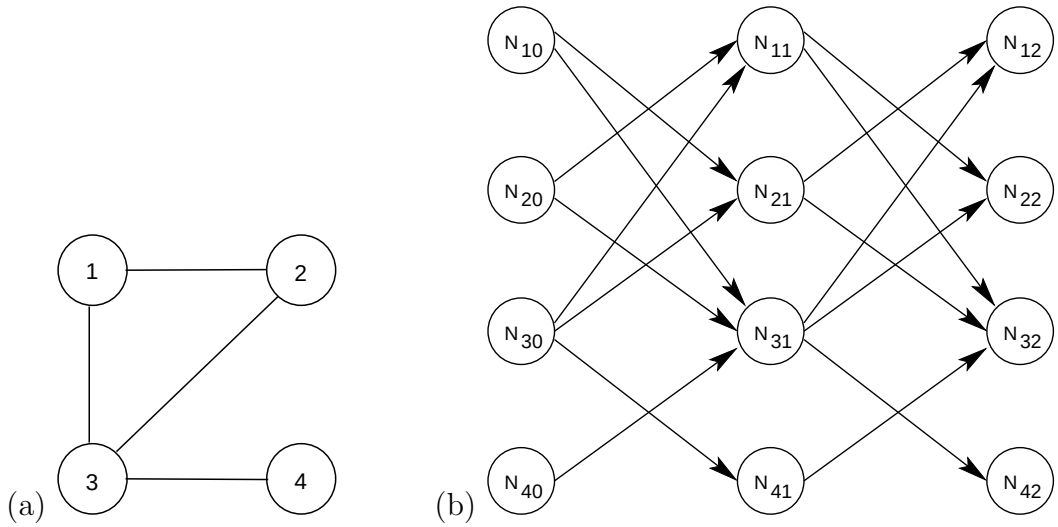


Figure 5-1. (a) An example graph (b) Time-expanded network for $T = 2$

formulation of the problem can be given as:

$$\text{maximize} \quad \sum_{i \in N} \hat{\gamma}_i y_i \tag{5–7}$$

$$\text{subject to} \quad \sum_{i \in N} x_i^t = 1 \quad \forall t = 0, \ldots, T \tag{5–8}$$

$$x_i^t \leq \sum_{j \in A(i)} x_j^{t-1} \quad \forall i \in N, t = 1, \ldots, T \tag{5–9}$$

$$y_i \leq \sum_{t=0}^{T} \sum_{j : i \in S(j)} x_j^t \quad \forall i \in N \tag{5–10}$$

$$0 \leq y_i \leq 1 \quad \forall i \in N \tag{5–11}$$

$$x_i^t \in \{0, 1\} \quad \forall i \in N, t = 0, \ldots, T. \tag{5–12}$$

Constraints (5–8) represent the fact that the searcher can visit only one node at a time. Constraints (5–9) ensure that node i can be visited at time t only if one of its neighbors has been visited at time $t - 1$. Constraints (5–10) force the value of y_i to zero unless node i can be observed by the searcher at some time period. Note that the y-variables will take on binary values in an optimal solution, and therefore we relax them as continuous variables. If the optimal objective function value of (5–7)–(5–12) is greater than 1, then

we have found a variable that has a negative reduced cost, and we therefore add the generated column to LHSLP.

5.2.2.2 Branch-and-price algorithm

The hide-and-seek problem can be solved using the following branch-and-price algorithm.

- Step 0: Choose a feasible set of initial search walks, in which every node is seen by at least one searcher.
- Step 1: Solve LHSLP, and generate columns by solving the searcher's problem until LHSLP has been solved to optimality. If the optimal solution is fractional, then branch, and go back to Step 1 for the subproblems. Else, stop processing the current subproblem with an integral solution.

We initialize our algorithm by generating a stationary searcher that stays at node i for T periods, for each $i \in N$. Even though these elementary searchers are not likely to be selected in an optimal solution, they guarantee the feasibility of LHSLP. We discuss several branching strategies that can be used for Step 1 in Section 5.5.

5.3 Pursuit Evasion Problem

In this section, we consider a pursuit evasion variant of the search problem. Unlike the hide-and-seek problem, the intruder is also mobile in this variant, and tries to avoid pursuit by the searchers. We note that this problem also reduces to the minimum dominating set problem for $S(i) = A(i),\ \forall i \in N$ and $T = 0$, and hence it is NP-hard.

5.3.1 Mathematical Model

Similar to the hide-and-seek problem, we define $P(T)$ to be the set of all possible walks of length at most T that can be taken by searchers. Similarly, let $R(T)$ denote the set of all possible walks of length $T + 1$ that can be taken by the intruder, thus potentially evading the searchers for more than T time periods. Let d_{pr} be a parameter whose value is 1 if a searcher following walk p detects an intruder following walk r, and 0 otherwise. This

problem can be formulated as a set covering problem, which we denote by PE:

$$\textbf{PE: minimize} \sum_{p \in P(T)} \lambda_p \tag{5–13}$$

$$\text{subject to} \sum_{p \in P(T)} d_{pr} \lambda_p \geq 1 \quad \forall r \in R(T) \tag{5–14}$$

$$\lambda_p \in \{0, 1\} \quad \forall p \in P(T), \tag{5–15}$$

where λ_p again equals 1 if and only if a searcher is assigned to follow walk p. The objective function (5–13) minimizes the number of searchers. Constraints (5–14) ensure that for each possible intruder walk of length $T + 1$, at least one searcher is selected to detect it.

5.3.2 Solution Approach

Once again, rather than enumerating all possible search patterns of length at most T, and all evasion patterns of length $T + 1$, we propose a dynamic column and row generation algorithm to solve the problem. We start with a subset of search patterns $P'(T) \subseteq P(T)$ and evasion patterns $R'(T) \subseteq R(T)$, and solve a limited pursuit evasion (LPE) problem, given $P'(T)$ and $R'(T)$. We next describe how we generate new search and evasion patterns as needed.

5.3.2.1 Searcher's problem

Given a subset $R'(T)$ of intruder walks, the searcher's problem is similar to the pricing problem in the hide-and-seek problem. Let γ_r be the dual variable associated with the constraint of type (5–14) corresponding to intruder walk $r \in R'(T)$. Also, let y_r be a decision variable that equals 1 if an intruder following walk r is detected by a searcher following a walk that we generate, and 0 otherwise, $\forall r \in R'(T)$. Given an optimal dual solution $\hat{\gamma}$ of the linear programming relaxation to LPE, we solve the following pricing problem to seek a λ-variable having a negative reduced cost: max $\sum_{r \in R'(T)} \hat{\gamma}_r y_r$, subject to the restriction that $(y_1, \ldots, y_{|R'(T)|})$ corresponds to a set of intruder walks detected by a searcher walk of length no more than T.

Similar to the previous pricing problem, this pricing problem can also be formulated as a mixed-integer programming problem on a time-expanded network consisting of $T+1$ stages. In particular, we create a node N_{it} for each $i \in N,\ t = 0, \ldots, T$. Unlike the previous problem, searchers do not necessarily have to move in each time period. Therefore, we connect each node to its neighbors and the copy of itself in the next stage. We define a binary variable x_i^t for all $i \in N,\ t = 0, \ldots, T$, which equals 1 if the searcher is located at node i at time t, and 0 otherwise. We also define a parameter $d_{ir}^t = 1$ if a searcher located at node i at time t can detect an intruder following walk r. An integer programming formulation of the pricing problem can be given as follows:

$$\text{maximize} \quad \sum_{r \in R'(T)} \hat{\gamma}_r y_r \tag{5–16}$$

$$\text{subject to} \quad \sum_{i \in N} x_i^t = 1 \quad \forall t = 0, \ldots, T \tag{5–17}$$

$$x_i^t \leq \sum_{j \in A(i)} x_j^{t-1} \quad \forall i \in N, t = 1, \ldots, T \tag{5–18}$$

$$y_r \leq \sum_{t=0}^{T} \sum_{i \in N} d_{ir}^t x_i^t \quad \forall r \in R'(T) \tag{5–19}$$

$$0 \leq y_r \leq 1 \quad \forall r \in R'(T) \tag{5–20}$$

$$x_i^t \in \{0, 1\} \quad \forall i \in N, t = 1, \ldots, T. \tag{5–21}$$

Constraints (5–17) ensure that the searcher cannot be located at multiple nodes simultaneously. Constraints (5–18) model the fact that the searcher can either stay at the same node, or can move to an adjacent node at each period. Constraints (5–19) represent the condition that the searcher detects intruder $r \in R'(T)$ only if it moves to a node where it can detect the intruder during the pursuit. We note that the y-variables can be relaxed as continuous variables in this case, too. This property allows the number of binary variables in the searcher's problem to stay constant as new evasion paths for the intruder are discovered. As before, if the optimal objective function value of (5–16)–(5–21) is greater than 1, then we have found a variable whose reduced cost is negative.

5.3.2.2 Intruder's problem

Given an integer feasible solution $\hat{\lambda}$ of LPE, in which a subset of the searchers has been selected, we need to solve a subproblem for the intruder to seek a walk that evades the searchers for more than T time units. To solve this problem, we generate a time-expanded network consisting of $T + 1$ stages, which contains a node N_{it} for each $i \in N,\ t = 0, \ldots, T$. We add a dummy start node s, and connect s to all nodes in the first stage, which corresponds to the initial intrusion at $t = 0$. Similar to the network generated for the searcher's problem, we connect each node to its neighbors and the copy of itself in the next stage. Finally, we connect all nodes in the last stage to a dummy node q. We then trace each selected searcher's walk, and eliminate the nodes (and the corresponding arcs) from the time-expanded network that would lead to the detection of the intruder.

After constructing the time-expanded network as described, we seek a feasible s–q path on the network by a standard breadth-first-search algorithm, which works in $O(N^2T)$ time in the worst case if G is dense. If such a path exists, then it corresponds to a walk r that the intruder can take to avoid detection for $T + 1$ time units. In this case, we add r to $R'(T)$, and generate the associated constraint of type (5–14). On the other hand, if no such path exists, then $\hat{\lambda}$ is a feasible solution of PA.

5.3.2.3 Branch-cut-price algorithm

The pursuit evasion problem can be solved using the following branch-cut-price algorithm.

- Step 0: Choose a set of initial search and evasion walks so that every node is seen by at least one searcher.
- Step 1: Solve the linear programming relaxation of LPE, and generate columns by solving the searcher's problem until linear programming relaxation of LPE has been solved to optimality. If all columns are integer-valued, go to Step 2. Else, branch, and go back to Step 1 for the subproblems.
- Step 2: Evade by solving the intruder's problem. If the intruder can evade the searchers, then add the evasion walk as a cutting plane of type (5–14), and go back to Step 1. Else, stop processing the current subproblem with an integral solution.

We initialize our algorithm by generating a stationary searcher that stays at node i for T periods, and a stationary evader that stays at node i for $T+1$ periods, for each $i \in N$. We discuss several branching strategies that can be used for Step 1 in Section 5.5.

5.4 Patrol Problem

5.4.1 Problem Description

The setting for the patrol problem that we consider in this section is as follows. The searchers are assigned to repeated patrol circuits, which they follow indefinitely. We assume that the period of a patrol circuit is bounded from above by a parameter K, where $K \geq 1$. Such a restriction might be due to a capacity or range limit of the searchers, or due to desired frequency of visits to individual nodes. Initially there is no intruder in the system. The intruder observes the searchers for a duration of time that is long enough to identify search patterns, and then picks a node and time to enter the system. It then tries to stay in the system as long as possible without being detected. The goal is to find the minimum number of searchers needed, along with the corresponding patrol routes, to ensure that the intruder is detected within T time periods after the intrusion. We note that this problem also reduces to the minimum dominating set problem (for $K = 1$, $S(i) = A(i),\ \forall i \in N$ and $T = 0$), and hence is also NP-hard.

5.4.2 Mathematical Model

Let $P^c(K)$ denote the set of all possible circuits of period no more than K that can be taken by searchers. Similarly, let $R(T)$ denote the set of all possible walks of length $T+1$ that can be taken by the intruder. Let d_{pr} be a parameter whose value is 1 if a searcher following circuit p detects an intruder following walk r, and 0 otherwise. Let us define a binary variable λ_p for all $p \in P^c(K)$, which equals 1 if a searcher is assigned to follow circuit p, and 0 otherwise. The patrol problem can be formulated as a set covering problem as follows.

$$\textbf{PP: minimize} \sum_{p \in P^c(K)} \lambda_p \tag{5–22}$$

$$\text{subject to} \sum_{p \in P^c(K)} d_{pr}\lambda_p \geq 1 \quad \forall r \in R(T) \tag{5–23}$$

$$\lambda_p \in \{0,1\} \quad \forall p \in P^c(K) \tag{5–24}$$

We propose a branch-cut-price algorithm similar to the pursuit evasion problem for solving this problem. We start with a subset of patrol routes $P'^c(K)$ and evasion walks $R'(T)$, and solve the resulting limited patrol problem (LPP). We generate new patrol routes and evasion walks as needed.

5.4.2.1 Searcher's problem

The searcher's problem is similar to the pricing problems discussed before. Let γ_r be the dual variable associated with the constraint of type (5–23) corresponding to intruder walk $r \in R'(T)$. We define y_r to be a decision variable that equals 1 if an intruder following walk r is detected by a searcher following a patrol circuit that we generate, and 0 otherwise, $\forall r \in R'(T)$. Given an optimal dual solution $\hat{\gamma}$ of the linear programming relaxation to LPP, we solve the following pricing problem to seek a λ-variable having a negative reduced cost: max $\sum_{r \in R'(T)} \hat{\gamma}_r y_r$, subject to the restriction that $(y_1, \ldots, y_{|R'(T)|})$ corresponds to a set of intruder walks detected by a searcher following a patrol circuit of period no more than K.

We can solve the searcher's problem by solving a series of integer programs as follows. Let τ denote the length of the current circuit under consideration. By considering different values of $\tau \in \{1, \ldots, K\}$ we can find a circuit that optimizes the searcher's problem. Note that some values of τ may not correspond to any circuits in G. For each value of τ, we generate a time-expanded network containing $\tau + 1$ levels, where the first level corresponds to the initial deployment of the searcher, and the last layer is a dummy layer that we use to model the recurring patrol patterns. We connect each node to its neighbors in the next stage. As before, we define a binary variable x_i^t for all $i \in N,\ t = 0, \ldots, \tau$, which equals 1 if the searcher is located at node i at time t. We also define a parameter $d_{ir}^t = 1$ if a searcher located at node i at time t can detect an intruder following walk $r \in R'(T)$. We

solve the following integer program to seek an optimal searcher circuit visiting exactly τ nodes.

$$\text{maximize} \quad \sum_{r \in R'(T)} \hat{\gamma}_r y_r \tag{5–25}$$

$$\text{subject to} \quad \sum_{i \in N} x_i^t = 1 \quad \forall t = 0, \ldots, \tau \tag{5–26}$$

$$x_i^t \leq \sum_{j \in A(i), j \neq i} x_j^{t-1} \quad \forall i \in N,\ t = 1, \ldots, \tau \tag{5–27}$$

$$x_i^0 = x_i^\tau \quad \forall i \in N \tag{5–28}$$

$$y_r \leq \sum_{t=0}^{\tau-1} \sum_{i \in N} d_{ir}^t x_i^t \quad \forall r \in R'(T) \tag{5–29}$$

$$0 \leq y_r \leq 1 \quad \forall r \in R'(T) \tag{5–30}$$

$$x_i^t \in \{0, 1\} \quad \forall i \in N,\ t = 0, \ldots, \tau. \tag{5–31}$$

Constraints (5–26) and (5–27) ensure that each feasible solution corresponds to a walk. Constraints (5–28) guarantee that the first and the last nodes visited by the searcher are the same, and hence the searcher's walk forms a circuit. Finally, Constraints (5–29) relate the x- and y-variables, where we can once again relax integrality restrictions on the y-variables.

Integer programs corresponding to different values of τ can be solved in any sequence. We note that a good solution obtained by solving the searcher problem for a particular value of τ can be used to prune problems to be solved later for different values of τ by bound. Therefore, we can start by solving a searcher problem for the largest value of τ, since a searcher following a longer circuit is more likely to detect more intruder walks. Also, we skip any $\tilde{\tau}$ if we determine that no circuit of length $\tilde{\tau}$ exists in G.

5.4.2.2 Intruder's problem

Given a selected subset of the searcher circuits, the intruder seeks a way of staying in the system as long as possible without being detected. Our main observation is that the

state of the system with respect to the location of the searchers is cyclic, and its period is equal to the least common multiple of the periods of searchers' circuits, which we denote by L. Therefore, the intruder can stay in the system indefinitely if it can identify a walk that allows it to return to its initial location at the end of a multiple of L steps.

To solve the intruder's problem, we can generate a time-expanded network consisting of L stages similar to the pursuit evasion problem. We connect each node to the copy of itself and its neighbors in the next stage by a directed arc having length 1. We also connect the nodes corresponding to stage L to the nodes to their neighbors in the first stage with directed arc having length 1 (modeling the fact that the overall search pattern repeats after L periods). We add a dummy start node s and a dummy end node q. We connect s to all nodes by a directed arc having length 0 (reflecting our assumption that the intruder can enter the system at any time and location), and connect all nodes to q by a directed arc having length 0. Finally, we trace each selected searcher's circuit, and remove nodes and arcs from the intruder's network that would lead to the detection of the intruder by the searcher.

We can solve the intruder's problem on the generated graph by seeking a longest s–q path. We first seek a topological ordering of the nodes using a standard depth-first-search algorithm, whose complexity is $O(N^2L)$ for a dense G. Since a directed graph is acyclic if and only if it is has a topological order, this step identifies whether the graph is cyclic. If there is a cycle in this graph, then the intruder can stay in the system forever without being detected by the searchers. In this case, we generate a cut of type (5–23), and stop. Otherwise, the graph is acyclic, and given a topological order of the nodes, a longest path can be found in polynomial time by a dynamic programming algorithm whose complexity is $O(N^2L)$ (Ahuja et al., 1993). If the length of a longest path is greater than T, then the intruder can successfully evade the searchers. We can use such a longest walk to generate a cut of type (5–23).

5.4.2.3 Branch-cut-price algorithm

The patrol problem can be solved using a branch-cut-price algorithm similar to the algorithm we propose for the pursuit evasion problem.

- Step 0: Choose an initial set of patrol circuits and evasion walks so that so that every node is seen by at least one searcher.
- Step 1: Solve the linear programming relaxation of LPP, and generate columns by solving the searcher's problem until the linear programming relaxation of LPP has been solved to optimality. If all columns are integer-valued, go to Step 2. Else, branch, and go back to Step 1 for the subproblems.
- Step 2: Evade by solving the intruder's problem. If the intruder can evade the searchers for more than T time periods, then add the evasion walk as a cutting plane of type (5–23), and go back to Step 1. Else, stop processing the current subproblem with an integer solution.

As before, we initialize our algorithm by generating a stationary searcher that stays at node i, and a stationary evader that stays in node i for $T + 1$ periods, for each $i \in N$. We discuss several branching strategies that can be used in Step 1 in the next section.

5.5 Branching Strategies

If an optimal solution of the linear programming relaxation of the master problem is integer feasible after all necessary variables and constraints have been added in the column- and row-generation phases, then we have found an optimal solution. Else, if some λ-variable is fractional, then we need to branch. Choosing a branching rule that does not make the pricing problem too difficult to solve is essential in column generation algorithms. Finding such a branching rule is relatively easy if the master problem is a set packing or set partitioning problem, since in this case it is possible to branch by choosing a subset of variables $\mathcal{X}$ and setting $x = 0,\ \forall x \in \mathcal{X}$ in one branch, and $x = 0,\ \forall x \notin \mathcal{X}$ in the other branch. These branching constraints can be added implicitly by removing the corresponding variables from the problem, and adjusting the pricing problem so that those variables are never generated in future iterations. Since no constraints are added explicitly, no new dual variables are created. Therefore, the structure of the pricing problem stays

the same throughout all nodes of the branch-and-bound tree (Barnhart et al., 1998; Jünger and Thienel, 2000).

However, our master problem is a set cover problem, and the approach described above is not directly applicable. Any constraint on a group of variables for our set cover problem would necessitate adding a branching constraint like $\sum_{x \in \mathcal{X}} x \leq \tilde{\lambda}$ in one branch, and $\sum_{x \in \mathcal{X}} x \geq \tilde{\lambda} + 1$ (for some suitable $\tilde{\lambda}$) in the other branch. Since these constraints cannot be added implicitly, we need to handle a new dual variable for each branching constraint in the subproblem.

As an alternative, we propose a multi-tiered branching rule. Given a fractional solution $\hat{\lambda}$, we first evaluate the value of each constraint expression $v_r = \sum_{p \in P'(T)} d_{pr} \hat{\lambda}_p$. If there exists a fractional v_r value for some $r \in R'(T)$, then we branch on the constraint (5–23) corresponding to r as follows. In the up-branch, we simply change the right-hand-side value of the constraint as $\lceil v_r \rceil$. On the down-branch, we set the upper bound of the constraint expression to $\lfloor v_r \rfloor$ (and convert it to an equality constraint if $\lfloor v_r \rfloor = 1$). Note that branching on a constraint in this manner does not introduce any new dual variables or constraints that need to be considered in any of the pricing problems. Another benefit of this branching scheme is the following. If we branch down on a constraint and obtain $\sum_{p \in P} d_{pr} \lambda_p = 1$, we can then use set-partition type of branching schemes on the corresponding subproblem. This allows us to branch on a group of λ-variables in the subsequent branches without destroying the pricing problem structure.

Note that sign of the dual variable γ_r can change after branching down on the constraint (5–23) corresponding to r, which makes $y_r = 0$ optimal regardless of the values of x-variables. Therefore, we need to add constraints that force y_r to 1 if the searcher's chosen path detects intruder r. Hence, we add constraints

$$y_i \geq x_j^t \quad \forall i \in N,\ j \in S(i),\ t = 0, \ldots, T \tag{5–32}$$

$$y_r \geq d_{ir}^t x_i^t \quad \forall r \in R'(T),\ i \in N,\ t = 0, \ldots, T \tag{5–33}$$

$$y_r \geq d_{ir}^t x_i^t \quad \forall r \in R'(T),\ i \in N,\ t = 0, \ldots, \tau - 1 \tag{5–34}$$

to the searcher's pricing problem for the hide-and-seek, pursuit evasion, and patrol problems, respectively.

In general it is possible to have a fractional solution $\hat{\lambda}$ for which all v-values are integral, and therefore the branching rule described above cannot be applied. In such cases we can apply a simple variable-based branching rule. If there is a variable $\lambda_{\hat{p}}$ whose value is fractional, we can simply create two branches with $\lambda_{\hat{p}} = 0$ and $\lambda_{\hat{p}} = 1$. In the down-branch, we simply eliminate the column corresponding to $\lambda_{\hat{p}}$ from the set covering formulation. In the up-branch we need to adjust the right-hand-side vector of our set covering problem before eliminating $\lambda_{\hat{p}}$. In either case, we need to adjust the pricing problems so that the same variable cannot be regenerated. Recall that the solution methods we propose for all three pricing problems are based on a time-expanded network formulation. Let us denote by $W(p)$ the set of time-expanded node indices corresponding to a searcher walk (or circuit) p. We can enforce the condition that a walk p is not generated again by adding the following constraint to the corresponding searcher's pricing problem:

$$\sum_{(i,t) \in W(p)} x_i^t \leq |W(p)| - 1. \tag{5–35}$$

Branching on a single variable is likely to be quite weak on the down-branch since only one particular searcher walk (or circuit) is avoided. Hence, we only apply this rule if our constraint-based branching rule fails. Also note that the difficulty of solving the set covering problem does not increase under either branching rule, since no constraints are added explicitly while branching.

5.6 Computational Results

We implemented the algorithms discussed in the previous sections in C++ on a Windows XP PC with a 3.4 GHz CPU and 2 GB RAM. We used CPLEX 11.2 to solve the

pricing problems and the linear programming relaxations of our set covering formulations. We implemented our algorithms for solving the intruder's problem using Boost Graph Library version 1.36 (Siek et al., 2001). Our base set of test problem instances consists of 150 randomly generated problem instances for which the expected edge density of the graph (measured as $\frac{|E|-|N|}{|N|\times(|N|-1)}$, where we do not consider self-loop edges in calculating edge density) is 10%, the number of nodes N ranges from 5 to 25, and the maximum allowed time to detection T ranges from 0 to 5. In generating instances we first picked a random subset of edges so that the edge density is 10%, and if necessary added the minimum number of edges needed to make the graph connected (see Siek et al. (2001)). We then added self-loop edges, and we generated five problem instances for each problem size, which is determined by the number of nodes. Finally, we solved each problem instance with different values of $T \in \{0, \ldots, 5\}$ for the hide-and-seek, pursuit evasion and patrol problems. In each case, we assume that a searcher located at node $i \in N$ can observe node i and all nodes adjacent to it, and hence we set $S(i) = A(i)$, for all $i \in N$. We imposed a 1200-second time limit past which we stopped the execution of an algorithm in all our experiments.

Recall that all problems that we consider in this chapter reduce to the minimum dominating set problem for $T = 0$. We use this property to calculate an initial upper bound by solving our LHS formulation, where we initialize $R'(0)$ by adding a searcher located at each node $i \in N$.

Our first experiment focuses on the hide-and-seek problem. For this experiment, we executed our branch-and-price algorithm described in Section 5.2.2.2 on our base data set. All 150 problem instances in our data set were solved to optimality within 12.8 seconds. Table 5-1 displays the average number of branch-and-bound nodes evaluated in our branch-and-price algorithm for different values of N and T. Each value represents the average of the results obtained on five randomly generated graphs. We observe that the number of branch-and-bound nodes that are explored increases as N increases, which

Table 5-1. Average number of branch-and-bound nodes explored for hide-and-seek problem

N	$T=0$	$T=1$	$T=2$	$T=3$	$T=4$	$T=5$
5	1	1	1	1	1	1
10	1	1.4	1.8	1.4	1.4	1
15	1	1	1	1	1	1
20	1	2.6	3.4	1.8	1.8	1
25	1	7.4	2.2	2.6	1.4	1

is expected since the difficulty of the problem increases with increasing N. Table 5-2

Table 5-2. Average number of searchers needed for the hide-and-seek problem

N	$T=0$	$T=1$	$T=2$	$T=3$	$T=4$	$T=5$
5	2	1.8	1	1	1	1
10	3.4	2.8	2	1.8	1.6	1
15	4.6	3.2	2.2	2	2	1.6
20	4.6	3.6	2.6	2	2	1.8
25	5.8	3.8	2.8	2	2	2

shows the average number of searchers needed for different values of N and T, where once again each value is an average of five problem instances. We note that the $T = 0$ column corresponds to the cardinality of the minimum dominating set. Table 5-2 reveals that the number of searchers needed increases as the graph gets larger, and decreases as the maximum allowed time to detect the intruder increases.

We analyze the performance of our branch-cut-price algorithm described in Section 5.3.2.3 for the pursuit evasion problem. Table 5-3 shows that our algorithm was able to solve 128 out of the 150 instances within the prescribed time limit. As expected, the difficulty of the problem increases as N and T increase, since this setting allows for more evasion routes for the intruder, and hence requires the searchers to develop more sophisticated routes.

Table 5-3. Number of instances that are solved within time limit for the pursuit evasion problem

N	$T=0$	$T=1$	$T=2$	$T=3$	$T=4$	$T=5$
5	5	5	5	5	5	5
10	5	5	5	5	5	5
15	5	5	5	5	5	5
20	5	5	5	3	1	2
25	5	5	4	3	0	0

Table 5-4 displays the average number of branch-and-bound nodes evaluated in our branch-cut-price algorithm for the pursuit evasion problem. We note that for this

Table 5-4. Average number of branch-and-bound nodes explored for the pursuit evasion problem

N	$T=0$	$T=1$	$T=2$	$T=3$	$T=4$	$T=5$
5	1	1	1	1	1	1
10	1	1.8	3	1.8	1.8	1
15	1	5	3.4	6.2	4.2	3
20	1	28.2	109	334.6	101	14.2
25	1	23.4	869	324.6	41.4	11.4

problem, processing each node takes longer than the hide-and-seek problem, and therefore our algorithm can process fewer nodes within the time limit. Finally, Table 5-5 displays the average number of searchers needed for this problem, where we use the best known solutions for instances that were not solved to optimality within the time limit. A

Table 5-5. Average number of searchers needed for the pursuit evasion problem

N	$T=0$	$T=1$	$T=2$	$T=3$	$T=4$	$T=5$
5	2	1.8	1	1	1	1
10	3.4	3	2.2	2	2	2
15	4.6	3.4	2.6	2	2	2
20	4.6	4	3	2.8	2.8	2.6
25	5.8	4	3.4	3.2	3.2	3

comparison of Tables 5-2 and 5-5 reveals that the number of searchers needed for the hide-and-seek problem is less than that for the pursuit evasion problem. This result is not surprising since the intruder is stationary in the former problem, while it can move to avoid the searchers in the latter problem.

Table 5-6. Number of instances that are solved within time limit for the patrol problem

N	$T=0$	$T=1$	$T=2$	$T=3$	$T=4$	$T=5$
5	5	5	5	5	5	5
10	5	5	5	5	5	5
15	5	5	5	5	5	5
20	5	1	1	1	1	0
25	5	0	0	0	0	0

Our last experiment evaluates our branch-cut-price algorithm described in Section 5.4.2.3 for the patrol problem. Table 5-6 reveals that our algorithm for the patrol problem can solve fewer instances in our data set within the time limit compared to our algorithms for the other problems. This can be explained by (i) our assumption that the intruder can pick a time to enter the system, and (ii) our solution algorithm for the searcher's problem, which requires solving multiple mixed-integer programs. The first factor makes it easier for

the intruder to evade the searchers, while the second factor makes the searcher's problem more difficult to solve. The combined effect of these factors is that processing each node takes longer than the other problems, which can also be seen by considering Table 5-7.

Table 5-7. Average number of branch-and-bound nodes explored for the patrol problem

N	$T=0$	$T=1$	$T=2$	$T=3$	$T=4$	$T=5$
5	1	1	1	1	1	1
10	1	1	1.4	1.4	1.4	1.4
15	1	44.6	20.2	6.2	11.4	3
20	1	12.2	5.4	4.2	3.4	1
25	1	5	3.2	2.1	1.2	1

Finally, we report our results on the number of searchers needed for the patrol problem in Table 5-8. As before, our calculations are based on the best known solutions and do not necessarily correspond to optimal solutions for the instances that were not solved within the time limit. However, we observe that the number of searchers needed for the patrol problem is larger than the other two problems as expected.

Table 5-8. Average number of searchers needed for the patrol problem

N	$T=0$	$T=1$	$T=2$	$T=3$	$T=4$	$T=5$
5	2	1.8	1.6	1.2	1	1
10	3.4	3	2.8	2.6	2.6	2.6
15	4.6	3.4	3	2.8	2.8	2.6
20	4.6	4.2	3.8	3.1	3	2.8
25	5.8	5.3	4.7	3.8	3.6	3.2

CHAPTER 6
CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

In this chapter, we focus on future research areas regarding the problems we have described in the previous chapters. We analyze the techniques we employed, identify associated weaknesses, and suggest improvements. We also discuss a research topic that is based on a technique for reformulating the master problem for a class of bi-level optimization problems.

6.1 Stochastic Edge-Partition Problem

For solving the stochastic edge-partition problem (Chapter 2, see also Taşkın et al. (2009a)) we first developed an integer programming formulation of the problem, and prescribed a bi-level reformulation of the problem that has integer variables in both stages. Our computational tests showed that both the direct solution of the integer programming formulation and our integer programming-based cutting plane algorithm for the bi-level formulation are capable of solving only small problem instances to optimality. We then designed a hybrid integer programming/constraint programming algorithm to overcome the computational difficulties encountered by the first two approaches. Our hybrid approach first allocates node copies that are to be distributed across configurations using an integer programming formulation, and then assigns nodes to subgraphs using a constraint programming algorithm. After assigning nodes to subgraphs, it partitions edges to subgraphs for each scenario in a third stage, using another integer programming formulation. Our computational experiments show that the hybrid approach significantly outperforms the other approaches.

In our study, we have assumed that the number of subgraphs, $|K|$, is a part of the problem input. In SONET network design application there is no practical limit on the number of subgraphs, but a limit is specified to model the problem (Goldschmidt et al., 2003; Sherali et al., 2000; Smith, 2005). Choosing $|K|$ too small may make the problem infeasible or suboptimal, and choosing $|K|$ too large increases the difficulty of the problem

as we discussed in Section 2.4. In our experiments, we manually chose $|K|$ sufficiently large to yield a feasible solution in each problem instance. An area for future research is to treat $|K|$ not as a parameter but as a variable, and to find the smallest possible value of $|K|$ that guarantees the existence of an optimal edge partition that minimizes the number of node copies. This problem appears to be very difficult in general, but some upper bounds can be derived for our problem. We first note that choosing $|K| = |E|$ is guaranteed to yield a feasible edge partition (and not eliminate any feasible edge partitions from consideration), since each edge can be partitioned into a unique subgraph. Furthermore, if a feasible edge partition having an objective function value $\hat{z}$ is known, then $\lfloor \hat{z}/2 \rfloor$ is an upper bound on $|K|$. This bound follows from the fact that there exists an optimal solution that contains at least two nodes in each non-empty subgraph. Such a $\hat{z}$ can be calculated by a simple improvement heuristic. We start with $|K| = |E|$, and initially assign each edge to a unique subgraph. We then seek two subgraphs that can be merged without violating any constraints in any scenario, while also improving the objective function value. If we find such subgraphs, we merge them and repeat this process until no more subgraphs can be merged. Since this algorithm starts with a feasible solution, and retains feasibility in each iteration, it yields a feasible solution. However, our preliminary analysis suggests that the bound on $|K|$ that we get using this approach is quite weak. Improving bounds on $|K|$ and finding the smallest possible number of subgraphs that yields an optimal edge partition is a future research area.

6.2 Matrix Decomposition Problem

In Chapter 3 we have described exact decomposition algorithms for solving leaf sequencing problems arising in IMRT treatment planning (also see Taşkın et al. (2009b)). Our solution algorithm for the matrix decomposition into apertures satisfying the consecutive-ones property is based on an integer programming model for finding a set of intensity values to be assigned to apertures, and a backtracking algorithm that forms apertures by finding compatible leaf positions for each row. Our computational results

show that an optimal solution to many clinical problem instances can be found within a few minutes. As such, not only can this algorithm reasonably be used in real clinical settings, but also the bounds obtained from our algorithm can serve as benchmark criteria to compare the performance of several heuristic methods prescribed for this problem.

Our solution technique for the consecutive-ones matrix decomposition problem and our three-stage approach to the stochastic edge-partition problem are based on a similar idea. In both problems, we add "aggregate variables" to our formulations, which describe important structural properties of solutions, but are not enough by themselves to encode complete solutions. In each case we represent the optimization problem in terms of our aggregate variables in a master problem, and provide a subproblem that seeks a complete feasible solution corresponding to the values of the aggregate variables chosen by the master problem. In both applications, our master problems are discrete optimization problems, which we solve using integer programming methods, and our subproblems are discrete feasibility problems, which we solve using constraint programming methods. Separating critical optimization decisions from feasibility decisions, and utilizing strengths of integer and constraint programming techniques in a hybrid algorithm has allowed us to obtain significantly better results than other methods. A major theme in our future research is going to be on generalizing this hybrid approach to handle a broader class of problems.

In Chapter 4 we studied a different variant of the matrix decomposition problem, in which the aperture matrices need to be rectangular in shape (see also Taşkın et al. (2008)). Rectangular apertures can be formed by using conventional jaws already integrated into IMRT treatment devices, and do not need an advanced MLC system, which is costly to manufacture and operate. In Chapter 4, we proposed an exact optimization algorithm that can be used to analyze whether a jaws-only treatment system can deliver fluence maps efficiently. Our algorithm is based on an integer programming

formulation, which we enhance using several valid inequalities and by partitioning the problem into simpler problems.

We also derived a bi-level Benders decomposition algorithm for this problem. The master problem of our Benders decomposition approach chooses a subset of the rectangular shapes that can be used in decomposing the input matrix. Later, a subproblem checks whether the selected subset of rectangles can completely decompose the input matrix. Unfortunately, our computational tests showed that our Benders decomposition algorithm is computationally inferior to the integer programming approach. The main reason of slow convergence is the weakness of cuts generated in each iteration. Specifically, given an infeasible subset of rectangles chosen by the master problem, our subproblem (which is a linear programming problem) detects infeasibility, and returns a cut, which is generated based on a dual extreme ray. However, this extreme ray is a *mathematical* proof of infeasibility, but does not necessarily identify the *underlying reason* of infeasibility. In other words, it does not identify which bixels in the input matrix cannot be partitioned with the selected subset of rectangles. Furthermore, there are typically many extreme dual rays for a single infeasible master problem solution, from which several non-dominated cuts can be derived. One way of improving the convergence can be applying a two-dimensional binary search algorithm on the input matrix to find out which region of the input matrix cannot be partitioned with the selected rectangles. That is, if the entire matrix cannot be partitioned, we try to partition the left-hand-side and the right-hand-side halves of the matrix independently. If one of these submatrices cannot be partitioned, we immediately have a more specific reason for infeasibility (and hence a stronger cut), because this result implies that the infeasibility in a submatrix needs to be fixed using a subset of the rectangles, which cover that part of the matrix. This idea can be applied recursively to find possibly multiple infeasible regions of the matrix. The cuts associated with these infeasible submatrices are much stronger than a single cut derived based on the entire matrix. Furthermore, the information regarding the

description of infeasibility generated by analyzing submatrices in a single iteration can only be retrieved after several iterations of the original decomposition algorithm, which is based on solving the entire matrix only. This idea of obtaining a better description of infeasibility by analyzing subsets of variables can be generalized to the general Benders decomposition algorithm, and has the potential of improving the convergence properties in many applications.

In our study, we have developed solution techniques for two versions of the matrix decomposition problem, which apply to most available IMRT treatment machinery. However, there are other types of machinery that have different aperture shape restrictions, such as interdigitation or connectedness constraints (see e.g., Lim (2002)). In a related line of research, we are planning to design exact optimization algorithms to solve other variants of the matrix decomposition problem to optimality. Quantifying the effect of several shape constraints enforced by different types of machinery on radiation delivery efficiency would be a valuable contribution to the medical physics field.

6.3 Graph Search Problem

In Chapter 5 we considered three variants of a graph search problem: (i) a hide-and-seek problem, where a set of searchers seek a stationary intruder, (ii) a pursuit evasion problem, where the intruder moves to avoid detection, and (iii) a patrol problem, where searchers follow recurring patrol routes. The aim in each problem is to find the minimum number of searchers needed so that the intruder cannot stay in the system without being detected for longer than a prespecified duration of time. We proposed a branch-cut-price algorithm, which can be adapted to all three problems with certain modifications. Our main contribution is that we do not make any assumptions on the topology of the input graph, and our algorithms work on general graphs.

Even though all three problems that we consider are NP-hard, our computational tests show that the hide-and-seek and pursuit evasion problems can be solved to optimality for modest problems sizes within reasonable computational time. However,

our algorithm for the patrol problem can only solve small problem instances to optimality within the limits that we enforced. Our future research will initially focus on improving the performance of our algorithm. In particular, we are planning to (i) add heuristics for the searcher's problem to seek variables having a negative reduced cost before switching to our mixed-integer programming models, (ii) investigate flow-based formulations for the searcher's problem, which might have a tighter linear programming relaxation than the node-based formulations that we proposed, and (iii) seek valid inequalities to improve dual bounds. We are also planning to extend our models and solution algorithms to handle different types of searchers with different capabilities and costs. For instance, some searchers might be able to detect the intruder from a longer distance, while some others might be stationary but cheaper to operate. A related problem that can be investigated is a setting in which we are given a fixed number of searchers, and which must be coordinated so that the amount of time an intruder can stay in the system is minimized.

Our set covering formulation, which is based on the idea that at least one searcher must be chosen for each route the intruder can take, can be generalized to other variants of the graph search problem. In particular, a problem that has been widely studied in the literature assumes that the intruder can reside at the edges of the graph. This problem has been investigated from a theoretical perspective (see, e.g., Dendris et al. (1997); Ellis et al. (1994); Seymour and Thomas (1993)); however, to the best of our knowledge no exact optimization method that works on general graphs has been proposed. We are planning to extend our algorithm so that it can also solve this problem.

6.4 Master Problem Reformulation in Bi-Level Cutting Plane Optimization Algorithms

In this line of research, we are planning to explore ways of reformulating master problems to improve the coordination between the master and subproblems in bi-level Benders optimization algorithms, resulting in faster convergence to an optimal solution.

The key concept behind this line of research is a surprising property of decomposition algorithms, namely the existence of several alternative optimal solutions (or extreme rays) to the dual of the second-stage problems, each resulting in a different Benders inequality. There can be an exponential number of these inequalities, each of which is nondominated. Worse, it is possible that each of these inequalities may need to be generated one at a time after each iteration of the master problem. However, it turns out that it is possible to reformulate the master problem to avoid this "exponential cut" difficulty in some cases. In a stochastic SONET design problem (Smith et al., 2004), and in a product introduction and interdiction game, Smith et al. (2008) consider the addition of a quadratic-order set of variables in the master problem. These new variables are passed to the subproblem, and a Benders cut is generated in terms of the new variables that implies all of the (exponentially-many) Benders cuts that could have been generated in the original variable space. This master problem reformulation technique has the potential to dramatically reduce the number of iterations required by Benders decomposition to converge to an optimal solution, with only a modest increase in the size of the formulation. In both cases mentioned above, the trade-off of increasing model size to improve the strength of Benders cuts was computationally beneficial.

Let us describe the idea in more detail in the context of a SONET network design problem (Smith et al., 2004), which is similar to the edge-partition problem discussed in Chapter 2. There exist a set of demand pairs $(i, j) \in E$ that may be satisfied on a single communications network (all demand pairs have to be satisfied in our edge-partition problem). The communications network is composed of "rings" $k = 1, \ldots, K$. If both clients i and j have been linked to ring k, then we may choose to satisfy the demand request between i and j on ring k. Define y^q_{ijk} to be a continuous variable that represents the fraction of the demand between i and j that is satisfied on ring k in scenario q (these variables are defined to be binary in our edge-partition problem since we assume that each demand pair needs to be satisfied on a single ring). Let E^q be the set of demand pairs

(i, j) in scenario q. We also define x_{ik} as a binary variable equal to one if and only if client i is assigned to ring k. Setting aside other issues in this problem such as ring capacity, a subset of scenario restrictions is:

$$y^q_{ijk} \leq x_{ik} \quad \forall q \in Q,\ (i,j) \in E^q,\ k \in K \tag{6–1}$$

$$y^q_{ijk} \leq x_{jk} \quad \forall q \in Q,\ (i,j) \in E^q,\ k \in K, \tag{6–2}$$

where the existence of scenarios $q \in Q$ is due to uncertain demands between clients i and j. Using a straightforward decomposition approach, the problem can be decomposed so that the x-variables are parts of the master problem formulation, while the y^q-variables are determined in subproblems corresponding to scenarios $q \in Q$. Constraints (6–1) and (6–2) essentially state that in order for the communication demand between customers i and j to be assigned to ring k, *both* customers i and j have to be assigned to ring k. Unfortunately, cuts enforcing this relationship cannot be represented in the original space of x-variables. Smith et al. (2004) show that there can exist an exponential number of alternative dual solutions associated with a master problem solution represented by $\hat{\mathbf{x}}$, each leading to a non-dominated Benders cut. Then they reformulate the master problem by adding u_{ijk} variables, which represent the minimum of x_{ik} and x_{jk}. In other words, $u_{ijk} = 1$ if both customers i and j are assigned to ring k. Given the u-variables, the constraints (6–1) and (6–2) can be replaced by

$$y^q_{ijk} \leq u_{ijk} \quad \forall q \in Q,\ (i,j) \in E^q,\ k \in K. \tag{6–3}$$

Smith et al. (2004) show that a single Benders cut based on the u-variables dominates an exponential number of cuts based on the original x-variables. In other words, adding a *quadratic* number of variables to the master problem can save an *exponential* number of iterations of the Benders decomposition algorithm. In this particular problem, the authors recognized that the y-variables in the subproblem are dependent on $\min(x_{ik}, x_{jk})$, and used this nonlinear relationship between the x-variables to reformulate the master

problem. This kind of relationship is quite common in bi-level optimization algorithms. Our goal in this line of research will be to generalize this approach to other types relationships between variables so that structures that can be exploited by master problem reformulation are identified automatically. We observe that such nonlinear relationships can be induced using suitable relaxations of the subproblem. For instance, assume that the subproblem contains two constraints like

$$a_1^1 y_1 + a_2^1 y_2 + \cdots + a_n^1 y_n \leq x_1 \tag{6–4}$$

$$a_1^2 y_1 + a_2^2 y_2 + \cdots + a_n^2 y_n \leq x_2, \tag{6–5}$$

where x_1 and x_2 are variables of the master problem. Since both (6–4) and (6–5) are of $\leq$ type, we can take the component-wise minimum of the two constraints to obtain

$$\min(a_1^1, a_1^2) y_1 + \min(a_1^1, a_1^2) y_2 + \cdots + \min(a_n^1, a_n^2) y_n \leq x_1 \tag{6–6}$$

$$\min(a_1^1, a_1^2) y_1 + \min(a_1^1, a_1^2) y_2 + \cdots + \min(a_n^1, a_n^2) y_n \leq x_2. \tag{6–7}$$

Since the left-hand-sides of (6–6) and (6–7) are the same, we can combine the two constraints into

$$\min(a_1^1, a_1^2) y_1 + \min(a_1^1, a_1^2) y_2 + \cdots + \min(a_n^1, a_n^2) y_n \leq \min(x_1, x_2). \tag{6–8}$$

Constraint (6–8) describes a nonlinear relationship between the y- and x-variables. At this point, the master problem can be reformulated by adding a variable $v_{12} = \min(x_1, x_2)$, and the subproblem can be reformulated by using this variable as

$$\min(a_1^1, a_1^2) y_1 + \min(a_1^1, a_1^2) y_2 + \cdots + \min(a_n^1, a_n^2) y_n \leq v_{12}. \tag{6–9}$$

Even though (6–9) is weaker than (6–4) and (6–5), this reformulation might improve the convergence of the algorithm due to the "exponential-cut" behavior, especially in the beginning iterations of the Benders process. In our future research we are planning

to formalize and generalize this idea, and determine how to apply this reformulation approach to improve computational performance in bi-level cutting plane algorithms.

APPENDIX A
AN IP MODEL FOR C1-MATRIX DECOMPOSITION PROBLEM

In this appendix we discuss an integer programming approach to decomposing a fluence map into a number of apertures and corresponding intensities that is based on a model proposed by Langer et al. (2001). Given a maximum number of unit-intensity apertures, say T, this formulation determines the positions of the left and right leaves in each row of each of these apertures. We develop the model by separately studying four components:

- *Fluence map requirements.* Define, for each aperture $t = 1, \ldots, T$ and each bixel $(i, j) \in \{1, \ldots, m\} \times \{1, \ldots, n\}$, a binary variable d_{ij}^t that is equal to one if and only if bixel (i, j) is exposed, i.e., not covered by a left leaf or a right leaf. Since each aperture has unit intensity, the following constraints then ensure that the desired fluence map is delivered:

$$\sum_{t=1}^{T} d_{ij}^t = b_{ij} \quad \forall\ i = 1, \ldots, m,\ j = 1, \ldots, n. \tag{A–1}$$

- *Aperture deliverability constraints.* Define, for each aperture $t = 1, \ldots, T$ and each bixel $(i, j) \in \{1, \ldots, m\} \times \{1, \ldots, n\}$, binary variables p_{ij}^t and l_{ij}^t that are equal to one if and only if bixel (i, j) is covered by the right leaf or the left leaf in row i of aperture t, respectively. The following set of constraints then ensure that each of the T apertures is deliverable:

$$\begin{aligned} p_{ij}^t + l_{ij}^t + d_{ij}^t &= 1 && \forall\ i = 1, \ldots, m,\ j = 1, \ldots, n,\ t = 1, \ldots, T && \text{(A–2)} \\ p_{ij}^t &\leq p_{i,j+1}^t && \forall\ i = 1, \ldots, m,\ j = 1, \ldots, n-1,\ t = 1, \ldots, T && \text{(A–3)} \\ l_{ij}^t &\leq l_{i,j-1}^t && \forall\ i = 1, \ldots, m,\ j = 2, \ldots, n,\ t = 1, \ldots, T. && \text{(A–4)} \end{aligned}$$

  In particular, constraints (A–2) state that each bixel is either covered by a right-hand leaf, covered by a left-hand leaf, or uncovered (where the d-variables are included only for convenience and can be substituted out of the formulation). Constraints (A–3) and (A–4) state that if any bixel (i, j) is covered by a right-hand leaf (resp. left-hand leaf), then bixel $(i, j+1)$ (resp. $(i, j-1)$) should be covered by a right-hand leaf (resp. left-hand leaf) as well.

- *Beam-on-time.* We associate a binary variable z^t with each aperture $t = 1, \ldots, T$ that is equal to one if there are uncovered bixels in aperture t and zero otherwise, so

that the beam-on-time is simply given by

$$\sum_{t=1}^{T} z^t. \tag{A–5}$$

While Langer et al. (2001) impose the following constraints to ensure that these variables have (at least) their desired value:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} d_{ij}^t \leq (mn) z^t \quad \forall\ t = 1, \ldots, T, \tag{A–6}$$

we note that the following stronger formulation, which would actually not require enforcing the z-variables to be binary, can be obtained by disaggregating (A–6).

$$d_{ij}^t \leq z^t \quad \forall\ i = 1, \ldots, m,\ j = 1, \ldots, n,\ t = 1, \ldots, T. \tag{A–6′}$$

Note that this model allows z^t to be equal to one even if in aperture t no bixels are exposed, so that formally speaking (A–5) is an upper bound on the beam-on-time. The objective function ensures that the z-variables take on their minimum possible value.

- *Number of apertures.* We associate a binary variable g_t with each aperture $t = 1, \ldots, T-1$ that is equal to one if aperture t is different from aperture $t+1$ and zero otherwise. The number of setups is then given by

$$\sum_{t=1}^{T} g_t. \tag{A–7}$$

(If any aperture is used more than once but separated by another one, we consider the second occurrence of the aperture to be a new setup. However, when minimizing total treatment time there always exists an optimal solution in which identical apertures are delivered sequentially.) Now let c_{ij}^t and u_{ij}^t be auxiliary binary variables such that the former is equal to one if bixel (i, j) is exposed in aperture t but not in aperture $t+1$ and zero otherwise, and the latter is equal to one if bixel (i, j) is covered in aperture t but not in aperture $t+1$. This relationship is stated by

$$-c_{ij}^t \leq d_{ij}^{t+1} - d_{ij}^t \leq u_{ij}^t \quad \forall\ i = 1, \ldots, m,\ j = 1, \ldots, n,\ t = 1, \ldots, T-1. \tag{A–8}$$

Langer et al. (2001) then use the following constraints to ensure that the variables g_t have (at least) their desired value:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} \left(c_{ij}^t + u_{ij}^t\right) \leq (mn) g^t \quad \forall\ t = 1, \ldots, T-1. \tag{A–9}$$

However, note that again a stronger set of inequalities (that permit g to be equivalently relaxed as continuous variables) is obtained using disaggregation:

$$c_{ij}^t + u_{ij}^t \leq g^t \quad \forall\ i = 1, \ldots, m,\ j = 1, \ldots, n,\ t = 1, \ldots, T-1. \tag{A–9′}$$

Similar to the case of the beam-on-time, this model allows g^t to be equal to one even if apertures t and $t+1$ are identical, although our objective function ensures that the g-variables are chosen sufficiently small.

Langer et al. (2001) then study the problem of minimizing the number of setups (A–7) subject to the constraints (A–2)–(A–4), (A–6), (A–9), the constraint that the beam-on-time is minimal:

$$\sum_{t=1}^{T} z^t \leq \tilde{z} \tag{A–10}$$

and binary constraints on the variables, where we recommend determining $\tilde{z}$ via one of the polynomial-time procedures mentioned in Section 3.1. We note that an equivalent model is obtained by simply setting $T = \tilde{z}$, which reduces the problem dimension, and hence should be more efficient than adding a beam-on-time constraint.

We wish to minimize the total treatment time as measured by

$$w_1 \sum_{t=1}^{T} g_t + w_2 \sum_{t=1}^{T} z^t \tag{A–11}$$

subject to constraints (A–2)–(A–4), (A–6′), (A–9′), and binary constraints on the appropriate variables (and hence we do not impose (A–10)).

APPENDIX B
COMPLEXITY OF C1-PARTITION

Proposition 3. C1-Partition *is strongly NP-complete.*

Proof. Let ξ be the subset of $\{1, \ldots, L\}$ such that $\ell \in \xi$ if and only if $x_\ell > 0$. Formally speaking, the problem size is given by $\log_2(L)$, n, and $|\xi|$ (since the zero entries of $\mathbf{x}$ need not be encoded).

Let $\mathcal{K}$ denote the set of all $O(n^2)$ n-dimensional binary vectors whose ones appear consecutively, where $\mathbf{v}_k$ is the binary vector corresponding to $k \in \mathcal{K}$. Consider a guessed solution that consists of $|\xi|$-dimensional nonnegative integer vectors $\mathbf{d}_k$, $\forall k \in \mathcal{K}$, where $d_{k\ell}$ denotes the number of times leaf position $\mathbf{v}_k$, $k \in \mathcal{K}$, is assigned to intensity $\ell \in \xi$. Since all $d_{k\ell} \leq L$ in some feasible solution, we restrict the guessed $\mathbf{d}$-vectors as such. The size of the guessed vectors is thus $O(n^2|\xi| \log_2(L))$. We can verify whether or not $\sum_{k\in\mathcal{K}} \sum_{\ell\in\xi} d_{k\ell}\mathbf{v}_k = \mathbf{b}$ in $O(n^2|\xi|)$ additions. Therefore, C1-Partition is in NP.

To show that C1-Partition is NP-complete, we reduce 3-Partition to it. 3-Partition is a strongly NP-complete problem and seeks whether a given multiset of integers can be partitioned into triplets having the same sum. Formally, it can be defined as follows (see Garey and Johnson (1979)):

3-Partition

INSTANCE: A multiset A of 3ν positive integers $a_1, \ldots, a_{3\nu}$ and a positive integer B such that $B/4 < a_i < B/2$ for $i = 1, \ldots, 3\nu$ and such that $\sum_{i=1}^{3\nu} a_i = \nu B$.

QUESTION: Can A be partitioned into ν disjoint multisets $A_1, \ldots, A_\nu$ such that $\sum_{j\in A_i} a_j = B$ for $i = 1, \ldots, \nu$?

Given an arbitrary instance of 3-Partition, we construct an instance for C1-Partition as follows. First, we define $\hat{x}$ to be an integer vector whose ℓ^{th} component, $\hat{x}_\ell$, is equal to the number of indices i for which $a_i = \ell$. Furthermore, we let $\mathbf{b}$ be a $(2\nu - 1)$-dimensional

vector of the form $[B \;\; 0 \;\; B \;\; 0 \;\; \cdots \;\; 0 \;\; B]$. We construct a feasible solution to C1-Partition that employs only the odd-indexed unit vectors of $\mathcal{K}$. Denote these vectors as $\mathbf{e}_1, \mathbf{e}_3, \ldots, \mathbf{e}_{2\nu-1}$, and index their associated $\mathbf{d}$-vectors as $\mathbf{d}_1, \mathbf{d}_3, \ldots, \mathbf{d}_{2\nu-1}$.

Assume that the 3-Partition instance is a yes-instance, and hence there exist multisets $A_1, \ldots, A_\nu$ such that $\sum_{j \in A_i} a_j = B$. In this case, a feasible solution of the C1-Partition instance lets $d_{2j-1,\ell}$ be the number of elements of intensity ℓ in A_j, for each $j = 1, \ldots, \nu$, and assigns $d_{k\ell} = 0$ for all other k. Similarly, suppose that the C1-Partition instance is a yes-instance. Since all positive values in $\mathbf{b}$ are adjacent to 0, in any feasible solution to the instance of C1-Partition, we may only use leaf positions that expose a single odd-index bixel. Also, since $B/4 < a_i < B/2, \; \forall \; i = 1, \ldots, 3\nu$, vector $\mathbf{d}_k$ must be used to deliver exactly three intensity values, for $k = 1, 3, \ldots, 2\nu - 1$. Then a feasible solution of the 3-Partition instance is given as multisets $A_1, \ldots, A_\nu$ recovered from $\mathbf{d}_1, \mathbf{d}_3, \ldots, \mathbf{d}_{2\nu-1}$ as described above. Therefore, an arbitrary 3-Partition instance is a yes-instance if and only if the corresponding transformed C1-Partition instance is a yes-instance. Since 3-Partition is strongly NP-complete, and since the transformation provided is polynomial in terms of the size of the problem and the instance data, it follows that C1-Partition is also strongly NP-complete. $\square$

REFERENCES

Agazaryan, N., T. D. Solberg. 2003. Segmental and dynamic intensity-modulated radiotherapy delivery techniques for micro-multileaf collimator. *Medical Physics* **30**(7) 1758–1765.

Ahuja, R. K., H. W. Hamacher. 2005. A network flow algorithm to minimize beam-on-time for unconstrained multileaf collimator problems in cancer radiation therapy. *Networks* **45**(1) 36–41.

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

Aigner, M., M. Fromme. 1984. A game of cops and robbers. *Discrete Applied Mathematics* **8**(1) 1–12.

Alpern, S. 1995. The rendezvous search problem. *SIAM Journal on Control and Optimization* **33**(3) 673–683.

Alpern, S. 2008. Hide-and-seek games on a tree to which Eulerian networks are attached. *Networks* **52**(3) 162–166.

Alpern, S., S. Gal. 2003. *The Theory of Search Games and Rendezvous*, *International Series in Operations Research & Management Science*, vol. 55. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Alspach, B. 2004. Searching and sweeping graphs: a brief survey. *Le Matematiche* **59** 5–37.

Alspach, B., D. Dyer, D. Hanson, B. Yang. 2008. Time constrained graph searching. *Theoretical Computer Science* **399**(3) 158–168.

Andreas, A. K., J. C. Smith. 2009. Decomposition algorithms for the design of a nonsimultaneous capacitated evacuation tree network. *Networks* **53**(2) 91–103.

Baatar, D. 2005. Matrix decomposition with time and cardinality objectives: Theory, algorithms, and application to multileaf collimator sequencing. Ph.D. thesis, Department of Mathematics, Technische Universität Kaiserslautern, Kaiserslautern, Germany.

Baatar, D., N. Boland, S. Brand, P. J. Stuckey. 2007. Minimum cardinality matrix decomposition into consecutive-ones matrices: CP and IP approaches. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Berlin, 1–15.

Baatar, D., H. W. Hamacher, M. Ehrgott, G. J. Woeginger. 2005. Decomposition of integer matrices and multileaf collimator sequencing. *Discrete Applied Mathematics* **152**(1) 6–34.

Barnhart, C., E. L. Johnson, G. L. Nemhauser., M. W. P. Savelsbergh, P. H. Vance. 1998. Branch-and-price: Column generation for solving huge integer programs. *Operations Research* **46**(3) 316–329.

Benders, J. F. 1962. Partitioning procedures for solving mixed variables programming problems. *Numerische Mathematik* **4** 238–252.

Benkoski, S. J., M.G. Monticino, J. R. Weisinger. 1991. A survey of the search theory literature. *Naval Research Logistics* **38** 469–494.

Bienstock, D., P. Seymour. 1991. Monotonicity in graph searching. *Journal of Algorithms* **12**(2) 239–245.

Bockmayr, A., N. Pisaruk. 2006. Detecting infeasibility and generating cuts for mixed integer programming using constraint programming. *Computers & Operations Research* **33**(10) 2777–2786.

Boland, N., H. W. Hamacher, F. Lenzen. 2004. Minimizing beam-on time in cancer radiation treatment using multileaf collimators. *Networks* **43**(4) 226–240.

Bortfeld, T. R., D. L. Kahler, T. J. Waldron, A. L. Boyer. 1994. X-ray field compensation with multileaf collimators. *International Journal of Radiation Oncology Biology Physics* **28**(3) 723–730.

Calafiore, G. C., M. C. Campi. 2005. Uncertain convex programs: Randomized solutions and confidence levels. *Mathematical Programming* **102** 25–46.

Carøe, C. C., R. Schultz. 1999. Dual decomposition in stochastic integer programming. *Operations Research Letters* **24** 37–45.

Chevaleyre, Y., F. Sempe, G. Ramalho. 2004. A theoretical analysis of multi-agent patrolling strategies. *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*. IEEE Computer Society, Washington, DC, USA, 1524–1525.

Codato, G., M. Fischetti. 2006. Combinatorial Benders' cuts for mixed-integer linear programming. *Operations Research* **54**(4) 756–766.

Dai, J., Y. Hu. 1999. Intensity-modulation radiotherapy using independent collimators: an algorithm study. *Medical Physics* **26**(12) 2562–2570.

Dai, J., Y. Zhu. 2001. Minimizing the number of segments in a delivery sequence for intensity-modulated radiation therapy with a multileaf collimator. *Medical Physics* **28**(10) 2113–2120.

Dendris, N. D., L. M. Kirousis, D. M. Thilikos. 1997. Fugitive-search games on graphs and related parameters. *Theoretical Computer Science* **172** 233–254.

Deng, J., T. Pawlicki, Y. Chen, J. Li, S. B. Jiang, C. M. Ma. 2001. The MLC tongue-and-groove effect on IMRT dose distributions. *Physics in Medicine and Biology* **46**(4) 1039–1060.

Earl, M. A., M. K. N. Afghan, C. X. Yu, Z. Jiang, D. M. Shepard. 2007. Jaws-only IMRT using direct aperture optimization. *Medical Physics* **34**(1) 307–314.

Ehrgott, M., H. W. Hamacher, M. Nußbaum. 2008. Decomposition of matrices and static multileaf collimators: A survey. C. J. S. Alves, P. M. Pardalos, L. N. Vicente, eds., *Optimization in Medicine*. Springer, New York, NY, 25–46.

Ellis, J. A., I. H. Sudborough, J. S. Turner. 1994. The vertex separation and search number of a graph. *Information and Computation* **113**(1) 50–79.

Engel, K. 2005. A new algorithm for optimal multileaf collimator field segmentation. *Discrete Applied Mathematics* **152**(1) 35–51.

Ernst, A. T., V. H. Mak, L. A. Mason. 2009. An exact method for the minimum cardinality problem in the planning of IMRT. *INFORMS Journal on Computing* Forthcoming.

Flocchini, P., M. J. Huang, F. L. Luccio. 2008. Decontamination of hypercubes by mobile agents. *Networks* **52**(3) 167–178.

Fomin, F. V., D. M. Thilikos. 2008. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science* **399**(3) 236–245.

Garey, M. R., D. S. Johnson. 1979. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., San Francisco, CA.

Goldschmidt, O., D. S. Hochbaum, A. Levin, E. V. Olinick. 2003. The SONET edge-partition problem. *Networks* **41** 13–23.

Goldstein, A. S., E. M. Reingold. 1995. The complexity of pursuit on a graph. *Theoretical Computer Science* **143**(1) 93–112.

Hahn, G. 2007. Cops, robbers and graphs. *Tatra Mountains Mathematical Publications* **36** 1–14.

Hamacher, H. W., K.-H. Küfer. 2002. Inverse radiation therapy planning – a multiple objective optimization approach. *Discrete Applied Mathematics* **118**(1) 145–161.

Haralick, R. M., G. L. Elliott. 1980. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence* **14**(3) 263–313.

Hooker, J. N. 2007. Planning and scheduling by logic-based Benders decomposition. *Operations Research* **55** 588–602.

Hooker, J. N., G. Ottosson. 2003. Logic-based Benders decomposition. *Mathematical Programming* **96** 33–60.

Huang, H. X., P. M. Pardalos. 2002. A multivariate partition approach to optimization problems. *Cybernetics and Systems Analysis* **38**(2) 265–275.

Isler, V., N. Karnad. 2008. The role of information in the cop-robber game. *Theoretical Computer Science* **399**(3) 179–190.

Jain, V., I. E. Grossmann. 2001. Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing* **13** 258–276.

Jotshi, A., R. Batta. 2008. Search for an immobile entity on a network. *European Journal of Operational Research* **191**(2) 347–359.

Jünger, M., S. Thienel. 2000. The ABACUS system for branch-and-cut-and-price algorithms in integer programming and combinatorial optimization. *Software Practice and Experience* **30**(11) 1325–1352.

Kalinowski, T. 2004. The algorithmic complexity of the minimization of the number of segments in multileaf collimator field segmentation. Tech. rep., Fachbereich Mathematik, Universität Rostock, Rostock, Germany.

Kalinowski, T. 2005a. A duality based algorithm for multileaf collimator field segmentation with interleaf collision constraint. *Discrete Applied Mathematics* **152**(1) 52–88.

Kalinowski, T. 2005b. Reducing the number of monitor units in multileaf collimator field segmentation. *Physics in Medicine and Biology* **50**(6) 1147–1161.

Kamath, S., S. Sahni, J. Li, J. Palta, S. Ranka. 2003. Leaf sequencing algorithms for segmented multileaf collimation. *Physics in Medicine and Biology* **48**(3) 307–324.

Kamath, S., S. Sahni, J. Palta, S. Ranka. 2004a. Algorithms for optimal sequencing of dynamic multileaf collimators. *Physics in Medicine and Biology* **49**(1) 33–54.

Kamath, S., S. Sahni, J. Palta, S. Ranka, J. Li. 2004b. Optimal leaf sequencing with elimination of tongue-and-groove underdosage. *Physics in Medicine and Biology* **49**(3) N7–N19.

Kamath, S., S. Sahni, S. Ranka, J. Li, J. Palta. 2004c. A comparison of step-and-shoot leaf sequencing algorithms that eliminate tongue-and-groove effects. *Physics in Medicine and Biology* **49**(14) 3137–3143.

Kamath, S., S. Sahni, S. Ranka, J. Li, J. Palta. 2004d. Optimal field splitting for large intensity-modulated fields. *Medical Physics* **31**(12) 3314–3323.

Kikuta, K., W. H. Ruckle. 2007. Rendezvous search on a star graph with examination costs. *European Journal of Operational Research* **181**(1) 298–304.

Kim, Y., L. J. Verhey, P. Xia. 2007. A feasibility study of using conventional jaws to deliver IMRT plans in the treatment of prostate cancer. *Physics in Medicine and Biology* **52**(8) 2147–2156.

Küfer, K.-H., M. Monz, A. Scherrer, H. L. Trinkaus, T. Bortfeld, C. Thieke. 2003. Intensity modulated radiotherapy – a large scale multi-criteria programming problem. *OR Spektrum* **25** 223–249.

Langer, M., V. Thai, L. Papiez. 2001. Improved leaf sequencing reduces segments or monitor units needed to deliver IMRT using multileaf collimators. *Medical Physics* **28**(12) 2450–2458.

LaPaugh, A. S. 1993. Recontamination does not help to search a graph. *Journal of the ACM* **40**(2) 224–245.

Laporte, G., F. V. Louveaux. 1993. The integer L-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters* **13** 133–142.

Lee, E. K., T. Fox, I. Crocker. 2000a. Optimization of radiosurgery treatment planning via mixed integer programming. *Medical Physics* **27**(5) 995–1004.

Lee, E. K., T. Fox, I. Crocker. 2003. Integer programming applied to intensity-modulated radiation treatment planning. *Annals of Operations Research* **119** 165–181.

Lee, Y., H. D. Sherali, J. Han, S. Kim. 2000b. A branch-and-cut algorithm for solving an intra-ring synchronous optical network design problem. *Networks* **35**(3) 223–232.

Lenzen, F. 2000. An integer programming approach to the multileaf collimator problem. Master's thesis, Department of Mathematics, University of Kaiserslautern, Kaiserslautern, Germany.

Lim, G. J. 2002. Optimization in radiation treatment planning. Ph.D. thesis, Computer Sciences Department, University of Wisconsin-Madison, Madison, WI.

Lim, G. J., J. Choi. 2007. A two-stage integer programming approach for optimizing leaf sequence in IMRT. Tech. rep., Department of Industrial Engineering, University of Houston, Houston, TX.

Lim, G. J., M. C. Ferris, D. M. Shepard. 2004. Optimization tools for radiation treatment planning in Matlab. M. L. Brandeau, F. Sainfort, W. P. Pierskalla, eds., *Operations Research and Health Care: A Handbook of Methods and Applications*. Kluwer Academic Publishers, Boston, MA, 775–806.

Lim, G. J., M. C. Ferris, S. J. Wright, D. M. Shepard, M. A. Earl. 2007. An optimization framework for conformal radiation treatment planning. *INFORMS Journal on Computing* **19**(3) 366–380.

Luedtke, J., S. Ahmed. 2008. A sample approximation approach for optimization with probabilistic constraints. *SIAM Journal on Optimization* **19** 674–699.

Lustig, I. J., J.-F. Puget. 2001. Program does not equal program: Constraint programming and its relationship to mathematical programming. *Interfaces* **31**(6) 29–53.

Marchand, H., A. Martin, R. Weismantel, L. Wolsey. 2002. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics* **123** 397–446.

Megiddo, N., S. L. Hakimi, M. R. Garey, D. S. Johnson, C. H. Papadimitriou. 1988. The complexity of searching a graph. *Journal of the ACM* **35**(1) 18–44.

Men, C., H. E. Romeijn, Z. C. Taşkın, J. F. Dempsey. 2007. An exact approach to direct aperture optimization in IMRT treatment planning. *Physics in Medicine and Biology* **52**(24) 7333–7352.

Migdalas, A., P. M. Pardalos. 1996. Hierarchical and bilevel programming. *Journal of Global Optimization* **8**(3).

Migdalas, A., P.M. Pardalos, P. Varbrand, eds. 1997. *Multilevel Optimization: Algorithms and Applications*. Kluwer Academic Publishers.

Nemhauser, G. L., L. A. Wolsey. 1988. *Integer and Combinatorial Optimization*. John Wiley & Sons, New York, NY.

Nemirovski, A., A. Shapiro. 2005. Scenario approximation of chance constraints. G. Calafiore, F. Dabbene, eds., *Probabilistic and Randomized Methods for Design Under Uncertainty*. Springer, London, 3–48.

Ntaimo, L., S. Sen. 2005. The million-variable "march" for stochastic combinatorial optimization. *Journal of Global Optimization* **32**(3) 385–400.

Parsons, T. D. 1978. Pursuit-evasion in a graph. *Theory and Applications of Graphs*. Springer, Berlin, 426–441.

Peng, S., C. Ho, T. Hsu, M. Ko, C. Y. Tang. 2000. Edge and node searching problems on trees. *Theoretical Computer Science* **240**(2) 429–446.

Penuel, J., J. C. Smith. 2009. Models and complexity analysis for the graph decontamination problem with mobile agents. Tech. rep., Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL.

Preciado-Walters, F., R. Rardin, M. Langer, V. Thai. 2004. A coupled column generation, mixed integer approach to optimal planning of intensity modulated radiation therapy for cancer. *Mathematical Programming* **101**(2) 319–338.

Que, W. 1999. Comparison of algorithms for multileaf collimator field segmentation. *Medical Physics* **26**(11) 2390–2396.

Que, W., J. Kung, J. Dai. 2004. "Tongue-and-groove" effect in intensity modulated radiotherapy with static multileaf collimator fields. *Physics in Medicine and Biology* **49**(3) 399–405.

Romeijn, H. E., R. K. Ahuja, J. F. Dempsey, A. Kumar. 2005. A column generation approach to radiation therapy treatment planning using aperture modulation. *SIAM Journal on Optimization* **15**(3) 838–862.

Romeijn, H. E., R. K. Ahuja, J. F. Dempsey, A. Kumar. 2006. A new linear programming approach to radiation therapy treatment planning problems. *Operations Research* **54**(2) 201–216.

Rossi, F., P. van Beek, T. Walsh, eds. 2006. *Handbook of Constraint Programming*. Elsevier.

Sak, T., J. W., S. K. Goldenstein. 2008. Probabilistic multiagent patrolling. *Advances in Artificial Intelligence – SBIA 2008*. Springer, Berlin, 124–133.

Sen, S., J. L. Higle. 2005. The C^3 theorem and a D^2 algorithm for large scale stochastic integer programming: Set convexification. *Mathematical Programming* **104** 1–20.

Seymour, P., R. Thomas. 1993. Graph searching and a min-max theorem for tree-width. *Jounal of Combinatorial Theory, Series B* **58**(1) 22–33.

Shepard, D. M., M. A. Earl, X. A. Li, S. Naqvi, C. Yu. 2002. Direct aperture optimization: a turnkey solution for step-and-shoot IMRT. *Medical Physics* **29**(6) 1007–1018.

Shepard, D. M., M. C. Ferris, G. H. Olivera, T. R. Mackie. 1999. Optimizing the delivery of radiation therapy to cancer patients. *SIAM Review* **41**(4) 721–744.

Sherali, H. D., B. M. P. Fraticelli. 2002. A modification of Benders' decomposition algorithm for discrete subproblems: an approach for stochastic programs with integer recourse. *Journal of Global Optimization* **22** 319–342.

Sherali, H. D., J. C. Smith. 2001. Improving zero-one model representations via symmetry considerations. *Management Science* **47**(10) 1396–1407.

Sherali, H. D., J. C. Smith, Y. Lee. 2000. Enhanced model representations for an intra-ring synchronous optical network design problem allowing demand splitting. *INFORMS Journal on Computing* **12**(4) 284–298.

Siek, J. G., L. Lee, A. Lumsdaine. 2001. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley.

Siochi, R. A. C. 1999. Minimizing static intensity modulation delivery time using an intensity solid paradigm. *International Journal of Radiation Oncology Biology Physics* **43**(3) 671–680.

Siochi, R. A. C. 2004. Modifications to the IMFAST leaf sequencing optimization algorithm. *Medical Physics* **31**(12) 3267–3278.

Siochi, R. A. C. 2007. Variable depth recursion algorithm for leaf sequencing. *Medical Physics* **34**(2) 664–672.

Smith, B. M. 1995. A tutorial on constraint programming. Tech. rep., School of Computing, University of Leeds, Leeds, UK.

Smith, B. M. 2005. Symmetry and search in a network design problem. Roman Barták, Michela Milano, eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer, Berlin, 336–350.

Smith, J. C. 2004. Algorithms for distributing telecommunication traffic on a multiple-ring SONET-based network. *European Journal of Operational Research* **154**(3) 659–672.

Smith, J. C., C. Lim, A. Alptekinoğlu. 2008. Optimal mixed-integer programming and heuristic methods for a bilevel Stackelberg product introduction game. Tech. rep., Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL.

Smith, J. C., A. J. Schaefer, J. W. Yen. 2004. A stochastic integer programming approach to solving a synchronous optical network ring design problem. *Networks* **44**(1) 12–26.

Stell, A. M., O. A. Zeidan, J. G. Li, J. F. Dempsey. 2004. An extensive log-file analysis of step-and-shoot IMRT subfield delivery errors. *International Journal of Radiation Oncology Biology Physics* **31**(6) 1593–1602.

Sutter, A., F. Vanderbeck, L. A. Wolsey. 1998. Optimal placement of add/drop multiplexers: Heuristic and exact algorithms. *Operations Research* **46**(5) 719–728.

Taşkın, Z. C., J. C. Smith, S. Ahmed, A. J. Schaefer. 2009a. Cutting plane algorithms for solving a stochastic edge-partition problem. *Discrete Optimization* Forthcoming.

Taşkın, Z. C., J. C. Smith, H. E. Romeijn. 2008. Mixed-integer programming techniques for decomposing IMRT fluence maps using rectangular apertures. Tech. rep., Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL.

Taşkın, Z. C., J. C. Smith, H. E. Romeijn, J. F. Dempsey. 2009b. Optimal multileaf collimator leaf sequencing in IMRT treatment planning. *Operations Research* Forthcoming.

Thorsteinsson, E. S. 2001. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. Toby Walsh, ed., *CP '01: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*. Springer, 16–30.

van Beek, P. 2006. Backtracking search algorithms. F. Rossi, P. van Beek, T. Walsh, eds., *Handbook of Constraint Programming*. Elsevier, 85–134.

Van Santvoort, J. P. C., B. J. M. Heijmen. 1996. Dynamic multileaf collimation without "tongue-and-groove" underdosage effects. *Physics in Medicine and Biology* **41**(10) 2091–2105.

Wang, Y., D. Beck, X. Jiang, R. Roussev. 2005. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. Tech. rep., Microsoft Research, Redmond, WA.

Wolsey, L A. 1998. *Integer Programming*. Wiley-Interscience, New York, NY.

Xia, P., L. J. Verhey. 1998. Multileaf collimator leaf sequencing algorithm for intensity modulated beams with multiple static segments. *Medical Physics* **25**(8) 1424–1434.

BIOGRAPHICAL SKETCH

Z. Caner Taşkın was born in Balıkesir, Turkey on September 8, 1981. He earned his B.S. and M.S. degrees in Industrial Engineering from Boğaziçi University, İstanbul in 2003 and 2005, respectively. Before starting his doctorate study, he worked for ICRON Technologies as a product consultant, where he took role in advanced planning and scheduling projects for customers in several industries including steel, automotive, electronics and glass manufacturing industries. He will finish his Ph.D. degree in Industrial and Systems Engineering at the University of Florida in August 2009. Following graduation, he will join Department of Industrial Engineering at Boğaziçi University as a faculty member.