OPTIMIZATION OF SLICE MODELS

By

Meta M. Voelker

A dissertation submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

(Mathematics and Computation in Engineering)

at the

UNIVERSITY OF WISCONSIN – MADISON

2002

Abstract

We consider solving large-scale mathematical programming problems quickly and efficiently as part of a larger system. By considering how the programming problems are integrated into their systems, we show how the overall solution process can become more efficient. Specifically, we consider three systems: slice modeling, model building and treatment planning. Slice modeling describes a system consisting of multiple closely-related programs, which vary primarily in their data. By incorporating the common structure and shared data into the solution process, each program can be solved more efficiently, resulting in significant time reductions. Model building describes a system which builds a prediction model. As an example, we consider building a likelihood basis pursuit model. To obtain a complete model, we must select parameters that minimize misclassification error. This results in a series of programs that must be solved, and reduces to a nonlinear slice model. Treatment planning describes a system for radiation therapy that consists of multiple individual treatments. We consider improving the overall therapy by incorporating errors from individual treatments into the system, and show how to achieve (approximate) solutions for real-life examples, again as slice models.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Professor Michael Ferris. Without his invaluable advice and support, this work would not have been possible. In addition, I would like to thank Professors Olvi Mangasarian, Stephen Robinson and Grace Wahba for their advice and suggestions. From the MaCE program, I would like to thank Professors Eric Bach, Paul Milewski and Marshall Slemrod for their encouragement and support from the very beginning.

Many people have helped me on various aspects of this project. I would like to thank Alex Meeraus for introducing me to DEA, and Steven Dirkse and Michael Bussieck for their help in implementing the GAMS/DEA slice interface. My thanks also go to Stefan Scholtes and John Walden for providing me with real-world DEA applications. I am indebted to Professor Murray Clayton for his time and suggestions on the DEA sensitivity analysis process. In addition, my thanks go to Hao Zhang, who provided me with the background for and helped me understand the likelihood basis pursuit model. I also thank Jinho Lim, who provided me with his planning code and helped me with the real-life planning example.

I am indebted to my parents, and Jason and Monica for their love, support, and encouragement. Eric, Teela and the monsters also deserve so much

more than I can give for their patience, understanding, love and support.

This material is based on research partially supported by the National Science Foundation under grants CCR-9972372 and ACI-0113051, the Air Force Office of Scientific Research under grant F49620-01-1-0040 and Microsoft Corporation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Mathematical programming techniques are used to build models that answer optimization questions. Often times, their results provide only a part of the answer for which the user is looking. In such cases, the mathematical programming models that the user defines provide a means to reaching the end, but not the end itself. The mathematical programs need to be *integrated* into a larger system in order to answer the complete question.

We are interested in examining the interaction between optimization models and the larger systems of which they are a part. In the cases that we consider, mathematical programs play a key role in achieving the final solution by providing partial answers or by providing insight into the system. One mathematical program on its own cannot provide the entire solution. Sometimes, a series of mathematical programs together are enough; sometimes, other models that make use of the solutions must be used.

The most important consideration in optimization within integrated systems is the effect that the mathematical programs have on the larger system. Because the mathematical programs influence the larger system, we

want their solutions to be obtained easily and efficiently. If the programming model must be solved multiple times in the system, we must be able to do multiple solves efficiently. We do this by maintaining program structure between solves and starting later solves from earlier solution points. If the programming model must consider large amounts of data, then the data must be used effectively so that the solver has easy access to them. We do this by considering how the data are used in the model and balancing data storage with model requirements.

Another consideration is generalization; that is, the ability of the mathematical programs within the system to deal with changes in and to the system. In this sense we are not interested in significant changes that result in new systems. Rather, we consider changes that are suggested by taking the solution process a step further. For example, we may be interested in considering alternate sources of data, or we may develop alternative methods to deal with a portion of the system.

We focus on three different systems: slice modeling (Chapter 2), model building (Chapter 3), and treatment planning (Chapter 4). All three systems require multiple solutions to mathematical models involving large amounts of data. By considering how the mathematical programs interact and influence their corresponding system, we can improve the efficiency of the mathematical programs and thus, the system as a whole.

1.1 Slice Modeling

The first system is one in which a series of mathematical programs must be solved in order to obtain the complete solution. We refer to this as the slice modeling system. Slice models are collections of mathematical programs with the same structure but different data. Further, in slice models, often the data elements are related: some or most of the data may stay the same between programs, and so the programs differ only in a few rows or columns.

If we consider all of the data for all of the programs at once, we can define a specific program by pulling out its appropriate "slice" of data from the full set. For the k-th slice program, this idea can be expressed as follows:

$$\begin{aligned} \min_{x} \quad & f^k(x) \\ \text{subject to} \quad & A^k x = b^k \\ & x \in \mathcal{X} \end{aligned} \tag{1.1}$$

where A^k represents the matrix of constraint coefficients which (along with right-hand-sides b^k) are unique to the k-th program. The set $\mathcal{X}$, on the other hand, represents the (core) constraints and program structure that remain constant between programs. Note that the set $\mathcal{X}$ can be very complicated, comprising other general constraints and possibly integrality conditions.

Many applications can be regarded as slice models. In Chapter 2, we consider Data Envelopment Analysis (DEA) and cross-validation models.

DEA models are used to evaluate the efficiency of production units. In a DEA model, we have multiple production units, each of which takes in multiple inputs to produce multiple outputs. The purpose of DEA is to provide a measure of relative efficiency for each unit. To do this, each unit is compared to the all of the others through the use of a mathematical program. One mathematical program must be solved for each production unit under consideration. Upon completion, the DEA model provides the user with an efficiency rating for each production unit analyzed. Thus, the collection of efficiency programs together answers the full question, and so forms the complete system.

Another example of slice modeling comes from cross-validation problems, which are used to measure generalization ability. In cross-validation problems, we are interested in training and testing a model. To determine how well a model works, we would like to evaluate it on test data. However, we are only provided with one set of data with which to both train and test the model. Thus, we must take some of the data for testing, leaving the remaining data for training. In order to use the (finite) data effectively, we make use of cross-validation: the model is tested multiple times, each time with a different testing set. The data not used for testing are used for training at each stage. Upon completion, we obtain test set accuracy measures for each set of test data used. Considering all of the measures provides us with an idea of how well the trained model will perform in general, on different

selections of data.

Modeling languages allow us to define slice models easily. Using parameters for the data, we only need to specify the program once. Then the program can be solved multiple times using different data by simply specifying which data to use for each solve. This has been done for Data Envelopment Analysis (DEA) models in [40, 85] by using a loop structure, inside of which the data are redefined and the solve is re-executed. By using subsets, we can further control which individual programs are actually solved. In addition, because the model is separate from the solver, the same program structure can be used for linear, mixed integer, or nonlinear programs and solved under different solvers.

However, using the loop structure of a modeling language inside a slice model definition results in multiple model generations and multiple solver calls. But, moving from program k to program $k+1$ in a slice model consists only of a change of the data "slice" — here, the matrix A^k and vector b^k. We exploit this fact to solve slice models much more efficiently: rather than starting anew after program k is solved, we *modify* that program to obtain program $k+1$. This approach eliminates the need to build the same structure over and over, and also enables us to use solution information from solve k in solve $k+1$ and thus improve overall solution time.

We implement this approach as an interface between GAMS and the

CPLEX callable library. The interface is general, allowing us to deal easily with different slice models (including linear, mixed integer and simple quadratic models). It also significantly outperforms slice models implemented using the looping constructs of the modeling system. Because of this, we are able to solve very large real-world problems which were previously difficult to solve within the general framework. In addition, with the efficiency we achieve under the interface, we can extend DEA results to confidence intervals. The procedure we use is computationally-intensive, requiring thousands of slice models to be solved, but is easily done under the slice interface.

1.2 Model Building

The second system we consider also involves the solution to multiple mathematical programs, but in this system the result is a complete model, not a series of measurements. Specifically, we consider a nonparametric penalized likelihood approach for model building called likelihood basis pursuit (LBP). The goal of LBP is to build a model for finding the probabilities of binary outcomes, given explanatory vectors. To do this, LBP makes use of a log likelihood model, penalized by basis pursuit terms.

In the process of building the LBP model, parameters must be selected that balance maximizing the likelihood with minimizing the penalty terms. We want to choose parameters that minimize misclassification. This is done

by selecting parameters from a grid search that minimize the generalized approximate cross validation (GACV) function. The GACV function, though, is not easily represented in compact form; its functional values can only be obtained by solving two instances of the model. This results in costly function evaluations.

To find appropriate parameters, often a grid search is employed. This approach requires the solutions to hundreds or thousands of instances of the model (depending upon the problem type). However, this reduces to a nonlinear slice model, since only the parameters (data) are being changed between solves. We show how employing slice modeling techniques, as discussed in Chapter 2, significantly improves the efficiency of individual solves and thus speeds up the grid search. In addition, we consider making use of derivative-free optimization algorithms for the parameter selection, replacing the grid search. We show how, by seeding the derivative-free algorithms with a coarse grid search, these algorithms can find better solutions with fewer function evaluations.

1.3 Treatment Planning

The final system we consider, treatment planning, is very different from the other two. In this system, we consider a radiotherapy treatment that is delivered as a series of smaller dosages over a period of time. Because the

full radiation dose is not delivered all at once, errors (such as mechanical difficulties or patient alignment) can occur, resulting in changes to the radiation dose. We consider how the radiation treatments can be adapted to deal with errors in the delivery process. To do this, we formulate a model of the day-to-day planning problem as a stochastic linear program and exhibit the gains that can be achieved by incorporating uncertainty about errors during treatment into the planning process. Due to size and time restrictions, the model becomes intractable for realistic instances. In these cases, we make use of neuro-dynamic programming to approximate the stochastic solution, enabling us to derive results from our models for realistic time periods. These results allow us to generate practical rules of thumb that can be implemented in current planning technologies. As an example of this, we apply the rule of thumb to actual patient data, using a planning tool from [65] to determine the actual dose to deliver at each time stage.

Here the full system is more complicated. It involves a planning tool, which determines the radiation dose to deliver; a delivery mechanism to actually deliver the dose; and a feedback system that informs us of the dose that was delivered. In building the rules of thumb, we assume no knowledge about these portions of the system. We only require the results of the feedback, that is, the actual dose that has been delivered at each stage. Currently, the mechanisms to determine actual dose delivered are fairly primitive, but new machines are being developed that can generate more accurate information

about the actual dose delivered, allowing a planner to compensate for errors in delivery.

1.4 Common Ties

Besides being systems whose solutions depend upon mathematical programming problems, the three examples we consider are also related by the types of programs that are used. In each system, multiple programs of the same structure are solved, and therefore each system makes use of slice modeling. For example, in the model building system, the parameter selection reduces to a nonlinear slice model since the same program must be solved multiple times but with different penalty parameters. In the treatment planning system, the replanning that is done at each stage to find the implementable dose to deliver requires only a data change (just update the current dose delivered). If the planning tool is implemented as a mathematical programming problem (which it is in our real-life example), then this becomes a slice model. Note that this slice model, though, depends upon data available only at execution time since we only realize errors at this time.

In addition, each system can be data-intensive. For example, DEA slice models may analyze many units, each with multiple inputs and multiple outputs. In model building, the size of the model built depends upon the amount

of explanatory data available, which can be large. In real-life treatment planning, we must deal with patient data in three-dimensions. Thus, the data used in these systems must be used efficiently to find effective solutions.

Finally, each system can be generalized. As pointed out above, the DEA slice model results can be generalized to confidence intervals under a procedure that requires thousands of standard DEA solves (this procedure is described in Section 2.3). Due to the number of solves required, this procedure is much more time consuming than the original DEA system, and thus benefits much more from our model improvements. In the model building system, the grid search for parameter selection can be replaced with derivative-free optimization methods, allowing for more efficient use of the programming problem solutions. In the treatment planning system, since we assume no knowledge about the planning tool in building the rules of thumb, alternate planning tools may be used.

1.5 Contributions

We make several contributions. For slice models, we provide a modeling language interface that exploits the structure of the problems while still allowing for *general* formulations. Any type of linear, mixed integer or simple quadratic slice model can use the slice interface and obtain significant time

improvements. The interface exploits the GAMS GDX [81] routines to provide all of the slice solutions to the user from within the original model file. This allows the user to perform additional analysis on the results, such as sensitivity analysis (for DEA problems) or testing set accuracy (for cross-validation problems).

For the likelihood basis pursuit approach, we provide a model implementation that not only solves the resulting programs, but does so quickly and efficiently. The time improvement allows for the parameter selection to be performed in reasonable time using a simple grid search. This also allows the model to be extended to consider more effects (as in the full model) or alternate data types (such as categorical data). In addition, we show how the grid search can be improved using derivative-free optimization methods. This application is useful not only for the likelihood basis pursuit approach, but also for other approaches that require the selection of appropriate parameters.

For the treatment planning system, we provide a means to incorporate errors into the planning process. This allows a planner to improve the treatment, delivering more of the planned dose to the target. The improved dose distribution can be determined either on-line using the results of the previous time stage to determine the current dose to deliver; or off-line using the rules of thumb from multiple examples to determine the entire treatment plan prior to delivering any dose.

The improvements we obtain in these examples show that considering

how optimization is integrated into a system is advantageous. For example, we can improve overall efficiency of the system, as we show in the slice modeling and model building systems. We can also offer improvements to the current system, as we show in the treatment planning system. Therefore, by improving optimization within the system, the entire system benefits, becoming more efficient and providing better solutions.

Chapter 2

Slice Modeling

In this chapter, we consider the system of slice modeling, which is built from a series of closely-related mathematical programming problems. This system is solved when the solutions for all of the programming problems are obtained. To speed up individual solves, we make use of the slice modeling approach: we maintain program structure and core data, and we start later solves from earlier solutions. We have implemented this approach in the slice interface for the modeling language GAMS. This allows us to combine the benefits of a general modeling system, namely allowing the user to define any type of model, with the benefits of exploiting the structure of the slice model. Using two examples (DEA and cross-validation models), we show how the slice interface significantly improves the overall solution efficiency.

To motivate the need for a general slice interface, we begin by discussing DEA models and show how these models can be regarded as slice models. Besides the standard CCR DEA model (named for Charnes et al. for their formulation in [19]), we also describe variations and extensions that show the importance of being able to define general models. Following that, we

describe the slice interface and its implementation in GAMS. To show the advantage of the slice interface, we provide two DEA examples, one for determining the efficiency of hospitals and one for determining fishing fleet capacity. Then we show how the efficiency we achieve under the slice interface enables us to extend the results of the hospital DEA model to confidence intervals. Finally, we present additional slice models in the form of cross-validation problems to demonstrate the generality of the slice interface.

2.1 Slice Models for DEA

To evaluate the efficiency of firms in an industry, Farrell [34] proposed a method in which individual firms are compared to an efficiency frontier. For firms with a single output, this efficiency frontier is a curve which represents the various combinations of input factors that an efficient firm could use to produce unit output. Extending to multiple outputs, the efficiency frontier becomes a surface.

Under Farrell's approach [34], a firm's efficiency rating depends upon where it lies in relation to the efficiency frontier. Thus, to make the comparison, the efficiency frontier must be known. Rather than relying on a theoretical function to define the frontier, Farrell instead used an empirical function defined by the best results observed in practice, according to his definition of technical efficiency. This empirical function estimates the frontier with

line segments (one output) or facets (multiple outputs) by taking the most conservative frontier that is consistent with the physically-attainable points (either actual firms or other points determined through production assumptions such as disposability or convexity). Once the frontier is determined, a firm's efficiency is then obtained by solving a series of matrix equations, which look for the intersection between the line segment connecting the firm to the origin and the efficiency frontier.

Reformulating Farrell's approach in linear programming language, Charnes et al. [19] coined the term Data Envelopment Analysis (DEA). DEA is a technique that evaluates the relative performance of a number of decision-making-units (DMUs). A DMU can be anything that takes in inputs and produces outputs; examples include firms, banks, and even schools. DEA focuses on identifying inefficient DMUs by evaluating the set of DMUs with respect to the "best" DMU. This "best" DMU can either be an actual DMU in the set or a composite created from attributes of multiple DMUs. In Farrell's terms, this "best" DMU lies on the efficiency frontier.

The real power behind DEA is its ability to deal with multiple inputs and multiple outputs, without requiring that these inputs and outputs be related in any functional form [5]. The first DEA application was to evaluate the efficiency of the educational program Program Follow Through [21]. In their evaluation, the inputs included the education level of the mother, the highest occupation of a family member, parental time spent counseling the

child, and the number of teachers on site; the outputs included reading test scores, mathematics test scores, and self-esteem measures. Although these data were difficult to relate to each other and weight directly for evaluation purposes in the traditional manner, they were easily evaluated under the DEA approach. Since then, the basic DEA formulations have been extended and modified for a variety of applications [28]. We first discuss the basic CCR model, which considers applications with constant returns to scale.

2.1.1 CCR Model: Constant Returns to Scale

To determine the relative performance of a DMU, we first define efficiency in terms of the inputs and outputs. We let the vector $Y_{\cdot,k}$ represent the set of outputs for the k-th DMU and the vector $X_{\cdot,k}$ represent the set of inputs. Then efficiency for the k-th DMU can be defined by

$$\text{efficiency} = \frac{\text{weighted sum of outputs}}{\text{weighted sum of inputs}} = \frac{u^T Y_{\cdot,k}}{v^T X_{\cdot,k}}.$$

Under DEA, the inputs (X) and outputs (Y) are data; the weights (u, v) are variables. For each DMU, a different set of weights may be used; this allows for different operational organization and/or different valuation of inputs and outputs [30]. Obviously, each DMU would want to choose its most favorable set of weights. This raises the questions of (1) determining each DMU's best weights, and (2) comparing DMUs effectively based on different choices of weights.

Before answering these questions, we make the following production assumption:

Assumption 2.1 *$X, Y \geq 0$ with $X_{\cdot,k} \neq 0, Y_{\cdot,k} \neq 0 \ \forall k$.*

Typically, X, Y are dense matrices. Then, to determine the most favorable set of weights for the k-th DMU, we define the general fractional CCR model by:

$$\begin{aligned} \max_{u,v} \quad & h = \frac{u^T Y_{\cdot,k}}{v^T X_{\cdot,k}} \\ \text{subject to} \quad & \frac{u^T Y_{\cdot,i}}{v^T X_{\cdot,i}} \leq 1 \ \forall i \\ & u, v \geq 0 \end{aligned} \tag{2.1}$$

In this model, we maximize the efficiency score of DMU k subject to the constraint that the efficiency scores for all DMUs are less than or equal to 1 (or 100% — maximum efficiency). In order to compare DMUs based on efficiency, a problem of this form must be solved for *each* DMU in the set.

Note that model (2.1) may not have a solution because the denominators may become zero and so quantities in the model may become undefined. However, if $(\bar{u}, \bar{v})$ solves (2.1), then so does $\gamma(\bar{u}, \bar{v})$ for any $\gamma > 0$. To exclude such multiple solutions, we normalize (2.1). Our choice of normalization is to force

$$v^T X_{\cdot,k} = 1.$$

Imposing this additional constraint eliminates the possibility of the zero solution, although it does not remove the issue of undefined values.

Rather than dealing with this difficulty directly, we follow the DEA literature and convert model (2.1) to its Charnes-Cooper linear programming version [18, 19]. In doing so, we obtain what is commonly referred to as the dual (CCR) DEA problem:

CCR Dual Input Orientation

$$\begin{aligned} \max_{u,v} \quad & u^T Y_{\cdot,k} \\ \text{subject to} \quad & v^T X_{\cdot,k} = 1 \\ & u^T Y \leq v^T X \\ & u, v \geq 0 \end{aligned} \tag{2.2}$$

In this model, we look to maximize efficiency by directly manipulating the weights u, v.

Taking the dual of (2.2), we obtain the primal (CCR) DEA problem:

CCR Primal Input Orientation

$$\begin{aligned} \min_{\theta,\lambda} \quad & \theta \\ \text{subject to} \quad & \theta X_{\cdot,k} \geq X\lambda \\ & Y\lambda \geq Y_{\cdot,k} \\ & \lambda \geq 0 \end{aligned} \tag{2.3}$$

Here, rather than manipulating the weights directly, we look for a composite DMU (with inputs $X\lambda$ and outputs $Y\lambda$) that produces just as much output as DMU k but that uses at most a fractional amount ($\theta X_{\cdot,k}$) of input. If $\theta = 1$, then the composite DMU takes the same amount of at least one input as DMU k, and so DMU k is considered (weak) efficient.

Theorem 2.2 *Models (2.2) and (2.3) have a solution. Further, if the objective value of this solution is denoted by* $\overline{\theta}$*, then* $0 < \overline{\theta} \leq 1$.

Proof: A feasible solution for the primal input model is $\theta = 1$, $\lambda_k = 1$ and $\lambda_i = 0$ for $i \neq k$.

A feasible solution for the dual input model can be obtained by choosing any $v \geq 0$ such that $v^T X_{\cdot,k} = 1$. Then, by our assumptions on the data, we have that $v^T X \geq 0$. Thus, choosing $u = 0$ will satisfy the second constraint. So, both the primal and dual problems are feasible and thus a solution exists.

Let $\overline{\theta}$ be the optimal objective value. Then, from primal feasibility, we have that $\overline{\theta} \leq 1$. From dual feasibility, we also have that $\overline{\theta} \geq 0$.

Now, assume that $\overline{\theta} = 0$. Since $X_{\cdot,i} \geq 0, X_{\cdot,i} \neq 0 \; \forall i$ from Assumption 2.1, we have that $\lambda = 0$ in problem (2.3). Then, problem (2.3) also tells us that

$$0 \geq Y_{\cdot,k},$$

which contradictions Assumption 2.1. Therefore, $\overline{\theta} > 0$. $\square$

For an output orientation, we note that a similar conversion can be made [18, Remark 1]. This results in the following dual model (named to match the dual input version [63]):

CCR Dual Output Orientation

$$\begin{aligned} \min_{u,v} \quad & v^T X_{\cdot,k} \\ \text{subject to} \quad & u^T Y_{\cdot,k} = 1 \\ & u^T Y \leq v^T X \\ & u, v \geq 0 \end{aligned} \tag{2.4}$$

As in the input-oriented version, we directly manipulate the weights to maximize efficiency.

Taking the dual of (2.4), we obtain the primal output-oriented model:

CCR Primal Output Orientation

$$\begin{aligned} \max_{\phi,\lambda} \quad & \phi \\ \text{subject to} \quad & \phi Y_{\cdot,k} \leq Y\lambda \\ & X\lambda \leq X_{\cdot,k} \\ & \lambda \geq 0 \end{aligned} \tag{2.5}$$

Similarly, we look for a composite DMU (with inputs $X\lambda$ and outputs $Y\lambda$) that takes in at most the same input as DMU k, but produces a multiple ($\phi Y_{\cdot,k}$) of the output. If $\phi = 1$, then the composite DMU produces the same amount of at least one output as DMU k, and so DMU k is considered (weak) efficient.

Theorem 2.3 *Models (2.5) and (2.4) have a solution. Further, if the objective value of this solution is denoted by $\overline{\phi}$, then $\overline{\phi} \geq 1$.*

Proof: A feasible solution for the primal output model is $\phi = 1$, $\lambda_k = 1$ and $\lambda_i = 0$ for $i \neq k$.

Note that the dual output model is also feasible: we can choose any $u \geq 0$ such that $u^T Y_{\cdot,k} = 1$; then, we can choose any $v \geq 0$ such that $u^T Y_{\cdot,i} \leq v^T X_{\cdot,i}$ $\forall i$ (this is possible under Assumption 2.1 because we can scale v up by any positive number). Thus, a solution exists.

Let $\overline{\phi}$ be the optimal objective value. From primal feasibility, we have that $\overline{\phi} \geq 1$. $\square$

2.1.2 Weak and Strong Efficiency

In the CCR models above, θ and ϕ give us the efficiency score for DMU k. However, this efficiency is measured by scaling every input or output by the same amount. Thus, we may find some units rated as efficient ($\overline{\theta} = 1$ or $\overline{\phi} = 1$) which still take in more of some inputs or produce less of some outputs than their corresponding optimal (composite) DMU. In the following discussion, we consider only the input models (2.2) and (2.3); related results are easily obtained for the output models (2.4) and (2.5) in a similar manner.

To illustrate this difficulty, we consider the following example from [20], which has two inputs and one output:

	DMU A	DMU B
$X_{1,\cdot}$	2	2
$X_{2,\cdot}$	6	5
$Y_{1,\cdot}$	1	1

Examining the data, we conclude that DMU A should be inefficient because, although it produces the same amount of output, it takes in more of the second input to do this: $X_{2,A} > X_{2,B}$. However, in solving either input

model for DMU A, we find that $\overline{\theta} = 1$. This occurs because both DMUs take in the same (efficient) amount of the first input ($X_{1,A} = X_{1,B}$), and so θ cannot be less than 1. Using only $\overline{\theta}$ or $\overline{\phi}$ for an efficiency measurement, as we have done here, is referred to as finding the *weak efficiency* scores: if $\overline{\theta}$ or $\overline{\phi}$ is 1, then the current DMU is efficient in at least one input/output, but we cannot guarantee the same for all of the other inputs/outputs.

To take into consideration *individual* inputs/outputs, we need measure *strong efficiency*. First, we consider the weights obtained from the dual input model: for DMU A in the above example, we found $u = 1$, $v_1 = 0.5$ and $v_2 = 0$. Clearly, with $v_2 = 0$, we are disregarding consideration of the second input — the very input in which the two DMUs vary and which shows DMU A to be inefficient. This implies that we need to force consideration of every input (and every output) in order to measure strong efficiency. Thus, in the dual input model, we need to add the constraints

$$u, v > 0 \quad \Rightarrow \quad u, v \geq \epsilon e$$

where e is a vector of 1's and $\epsilon > 0$.

Adding this constraint to the dual input model has the effect of adding an additional objective function term involving the slack variables to the primal

input model:

$$\begin{array}{ll} \text{Primal Input Orientation for Strong Efficiency} & \\ \min\limits_{\theta,\lambda,s,t} & \theta - \epsilon(e^T s + e^T t) \\ \text{subject to} & \theta X_{\cdot,k} - s = X\lambda \\ & Y\lambda - t = Y_{\cdot,k} \\ & \lambda, s, t \geq 0 \end{array} \tag{2.6}$$

In this modified primal model, besides minimizing θ, we also look to maximize the sum of the slack variables s and t. Each slack variable tells us by how much the scaled input or output for the current DMU varies from the optimal (composite) DMU values. By maximizing the sum of the slack variables, we are looking to maximize the difference between the scaled inputs and outputs, and their efficient values. If the current DMU is only weak efficient (varying from the optimal DMU in some input or output), then the objective function slack sum is positive and reduces the efficiency rating of 1.

We now show that model (2.6) is valid for some ϵ:

Theorem 2.4 *The strong efficiency model (2.6) has a solution for some $\epsilon > 0$. Further, this solution also solves the weak efficiency model (2.3).*

Proof: We consider model (2.6) to be a perturbed version of (2.3). Let $\overline{\theta}$ be the minimum value for the original (unperturbed) problem and consider the

related problem:

$$\begin{aligned} \min_{\theta,\lambda,s,t} \quad & -(e^T s + e^T t) \\ \text{subject to} \quad & \theta X_{\cdot,k} - s = X\lambda \\ & Y\lambda - t = Y_{\cdot,k} \\ & \theta \leq \overline{\theta} \\ & \lambda, s, t \geq 0 \end{aligned} \tag{2.7}$$

Model (2.7) is primal feasible: simply choose the solution from the primal input model that gave $\overline{\theta}$.

Now, consider the corresponding dual of (2.7):

$$\begin{aligned} \max_{u,v,\gamma} \quad & u^T Y_{\cdot,k} - \overline{\theta}\gamma \\ \text{subject to} \quad & v^T X_{\cdot,k} - \gamma = 0 \\ & u^T Y \leq v^T X \\ & u, v \geq e \\ & \gamma \geq 0 \end{aligned} \tag{2.8}$$

Since $X, Y \geq 0$ and $X_{\cdot,i}, Y_{\cdot,i} \neq 0 \; \forall i$ by Assumption 2.1, we can choose $u, v > 0$ to satisfy the second constraint; scaling these values by some positive constant yields $u, v \geq e$ to satisfy the third constraint. Further, under these

values for v, we have (from the first constraint) that $\gamma = v^T X_{\cdot,k} \geq 0$, satisfying the fourth constraint. Thus, model (2.8) is feasible and so model (2.7) is also dual feasible.

Thus, model (2.7) has a solution. This satisfies the condition of Theorem 1 from [76]. Applying this, we have that there exists $(\theta^*, \lambda^*, s^*, t^*)$ and an ϵ^* such that $(\theta^*, \lambda^*, s^*, t^*)$ solves both the original problem from (2.3) and the perturbed problem (2.6) for $\epsilon \in [0, \epsilon^*]$. $\square$

The difficulty in using this approach for strong efficiency, though, is in finding a correct value for ϵ. As is noted in [1], choosing a good value for ϵ is difficult: making ϵ too large can result in an unbounded linear program; making ϵ too small can result in reduced costs close to the pricing tolerance, eliminating valid variables from the basis in actual computations. Rather than choosing a particular value for ϵ, the authors of [1] prefer to let ϵ represent the "non-Archimedean infinitesimal", the smallest positive number possible, which has no numeric value. Under their interpretation, we are unable to solve model (2.6) directly.

Instead, we use a two-phase approach as suggested in [28]. In phase 1, we solve the original (unperturbed) primal input model (2.3) to obtain the (weak) efficiency rating $\overline{\theta}$. In phase 2, we consider the efficiency of *individual* inputs and outputs. This is done by maximizing the difference between the

scaled inputs and outputs, and their most efficient values:

$$\begin{array}{ll} \text{Phase 2 Input Orientation} & \\ \min\limits_{\lambda,s,t} & -(e^T s + e^T t) \\ \text{subject to} & \overline{\theta} X_{\cdot,k} - s = X\lambda \\ & Y\lambda - t = Y_{\cdot,k} \\ & \lambda, s, t \geq 0 \end{array} \tag{2.9}$$

Notice that the constraints for the phase 2 problem are the same as the phase 1 problem, except that we have added explicit slack variables and replaced θ with its optimal value $\overline{\theta}$. Thus, any solution obtained from the phase 2 problem is also a solution of the phase 1 problem with $\theta = \overline{\theta}$. Essentially, in the phase 2 model, we are restricting our search to only those composite DMUs that are efficient with respect to the current DMU.

Theorem 2.5 *The two-phase approach is equivalent to solving the strong efficiency model (2.6) for sufficiently small* $\epsilon \geq 0$.

Proof: Solving the phase 1 problem yields the optimal value $\overline{\theta}$. For the phase 2 problem, we set $\theta = \overline{\theta}$ in the constraints of the phase 1 problem. This is equivalent to optimizing over the optimal solution set of the phase 1 problem. Since everything is linear, the perturbed portion of the objective function in model (2.6) is convex, and we can apply Theorem 3 from [76].

Doing so enables us to conclude that for sufficiently small $\epsilon \geq 0$, the solution of the two-phase approach is also a solution of the (perturbed) strong efficient problem. $\square$

2.1.3 CCR Models as Slice Models

The basic CCR models and the phase 2 model (2.9) fit the definition of slice models: a complete model consists of multiple linear programs (one for each DMU), each of which has the same structure but different data. In fact, these models are even more closely related than just structurally — a core set of data remains the same in each program.

For the basic dual CCR model (2.2), only one row and the objective function change (resulting from the condition that all of the efficiency scores must be less than or equal to 1); in terms of model (1.1), this implies that

$$f^k(u, v, \theta) = \theta,\ A^k = \begin{bmatrix} Y_{\cdot,k}^T & 0 & -1 \\ 0 & X_{\cdot,k}^T & 0 \end{bmatrix},\ b^k = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

and

$$\mathcal{X} = \left\{ \begin{pmatrix} u \\ v \\ \theta \end{pmatrix} \middle| \, u^T Y \leq v^T X; u, v \geq 0 \right\}.$$

For the primal model (2.3), one column and the right-hand-side change; in terms of model (1.1), this implies that

$$f^k(\lambda, s_x, s_y, \theta) = \theta,\ A^k = \begin{bmatrix} X & I & 0 & -X_{\cdot,k} \\ Y & 0 & -I & 0 \end{bmatrix},\ b^k = \begin{bmatrix} 0 \\ Y_{\cdot,k} \end{bmatrix} \tag{2.10}$$

and

$$\mathcal{X} = \left\{ \left(\begin{array}{c} \lambda \\ s_x \\ s_y \\ \theta \end{array} \right) \middle| \lambda, s_x, s_y \geq 0 \right\}. \tag{2.11}$$

Here, the slices contain most of the data and the core contains very little. If we introduce two auxiliary variables for $X\lambda$ and $Y\lambda$, then (2.10) and (2.11) become:

$$\begin{gathered} f^k(\lambda, s_x, s_y, \theta, f_x, f_y) = \theta, \\ A^k = \begin{bmatrix} 0 & I & 0 & -X_{\cdot,k} & I & 0 \\ 0 & 0 & -I & 0 & 0 & I \end{bmatrix}, \\ b^k = \begin{bmatrix} 0 \\ Y_{\cdot,k} \end{bmatrix} \end{gathered} \tag{2.12}$$

$$\mathcal{X} = \left\{ \left(\begin{array}{c} \lambda \\ s_x \\ s_y \\ \theta \\ f_x \\ f_y \end{array} \right) \middle| \begin{array}{c} f_x = X\lambda; f_y = Y\lambda \\ \\ \lambda, s_x, s_y \geq 0 \end{array} \right\}. \tag{2.13}$$

The data in each slice are reduced to $3(|i|+|o|)$ nonzeros, where i is the set of inputs and o is the set of outputs. The core part of the model then contains all the significant data.

A more natural slice formulation for the primal model (2.3) would be one where the slicing was done over the columns rather than the rows. This would eliminate the reformulation given in equations (2.12) and (2.13). However, we use the latter reformulation due to the way in which the modeling language defines models; in GAMS, the model is given in terms of the rows (or equations, in GAMS terminology) [17]. As we show later in Section 2.2.1, this formulation preference results in very little change to the model.

The phase 2 model (2.9) can be reformulated as a slice model in a similar fashion.

Because many real-world DEA problems have numerous DMUs and multiple inputs and outputs, the evaluation process can be very data-intensive and thus very time consuming under general linear programming solvers. Thinking of CCR models as slice models enables us to keep the core data constant between individual problems. Defining individual programs then consists of just adding specific data. This approach reduces problem generation time, and also allows us to easily include previous solution information (such as basic rows or columns) for the core data.

Besides solving larger real-world DEA problems, slice modeling also aids in extending DEA results to confidence intervals. Recent research by Simar and Wilson [97, 98, 99] finds confidence intervals for the DEA efficiency scores by repeatedly resampling from the original data set. Under their research, a smooth bootstrap is used to build an estimate for the probability density function of the original DEA efficiency scores. From this estimate, new efficiency scores are drawn and are used to obtain corresponding input-output data. Using this new data, the DEA problem for the DMUs is re-solved. Repeatedly applying this approach results in sampling distributions for the DMU's efficiency scores, from which confidence intervals can be drawn. To obtain accurate confidence intervals, a thousand or more samples may need to be drawn [99], requiring thousands of individual solves. Improving the efficiency of the individual solves and speeding up each DEA evaluation through

slice modeling enables us to implement this process and obtain decent confidence intervals. More details on the process and its implementation under a modeling language employing slice modeling, as well as results for a real-world example, are given in Section 2.3.

2.1.4 Other DEA Models

The CCR model is not the only DEA model that exists. Other DEA models that address application issues have been developed and used in practice. Some of these models are simple modifications of the CCR model; others vary more. With the variety of models available, each addressing different needs, it becomes important to allow for the definition of general DEA models.

In this section, we discuss other DEA models. After the CCR model, the BCC and additive models are the most general. While the CCR model deals with constant returns to scale (CRS), the BCC model deals with variable returns to scale (VRS). In addition, like the CCR model, the BCC model can take on an input orientation or an output orientation. The additive model differs from the CCR and BCC models in that it can deal with applications involving either CRS or VRS. Further unlike the CCR or BCC models, the additive model does not have an orientation but instead adjusts both the weighted input and the weighted output together. Besides the BCC and additive models, other variations and extensions exist that provide more specialized information or deal with specific applications.

BCC Model: Variable Returns to Scale

The difference between the CCR model and the BCC model (named for Banker et al. for their formulation in [6]) is the consideration of variable returns to scale — the assumption that the amount produced has an effect on the efficiency [5]. In this model, we restrict attention to determining efficiency based on the *current* level of operations [6].

The effect that this has on the linear programming model is to restrict our search for a more efficient composite DMU to the convex hull of the given DMUs by the addition of the constraint $e^T\lambda = 1$.

$$\begin{array}{llll} \text{BCC Primal Input Orientation} & & \text{BCC Dual Input Orientation} & \\ \min\limits_{\theta,\lambda} & \theta & \max\limits_{u,v,\mu} & u^T Y_{\cdot,k} + \mu \\ \text{subject to} & \theta X_{\cdot,k} \geq X\lambda & \text{subject to} & v^T X_{\cdot,k} = 1 \\ & Y\lambda \geq Y_{\cdot,k} & & u^T Y + \mu e \leq v^T X \\ & e^T\lambda = 1 & & u, v \geq 0 \\ & \lambda \geq 0 & & \mu \text{ free} \end{array} \tag{2.14}$$

$$
\begin{array}{lll|lll}
\multicolumn{3}{l|}{\text{BCC Primal Output Orientation}} & \multicolumn{3}{l}{\text{BCC Dual Output Orientation}} \\
\max\limits_{\phi,\lambda} & \phi & & \min\limits_{u,v,\nu} & v^T X_{\cdot,k} + \nu \\
\text{subject to} & \phi Y_{\cdot,k} \leq Y\lambda & & \text{subject to} & u^T Y_{\cdot,k} = 1 \\
& X\lambda \leq X_{\cdot,k} & & & u^T Y + \nu e \leq v^T X \\
& e^T\lambda = 1 & & & u, v \geq 0 \\
& \lambda \geq 0 & & & \nu \text{ free}
\end{array}
\tag{2.15}
$$

These models find weak efficiency; to find strong efficiency, a phase 2 problem with the additional constraint $e^T\lambda = 1$ can be solved.

Additive Model

The third most general DEA model is the additive model. The additive model does not have an orientation because it does not look to maximize the numerator or minimize the denominator of h in model (2.1); rather, it looks to maximize the difference between the weighted output and the weighted input. It is subject to the same constraints, namely that every DMU's weighted efficiency must be less than or equal to 1. However, rather than restricting the weights to be nonnegative, the weights are "normalized"

and restricted to be greater than or equal to 1.

$$\begin{array}{llll} \text{Additive Primal CRS} & & \text{Additive Dual CRS} & \\ \min\limits_{\lambda,s,t} & -(e^T s + e^T t) & \max\limits_{u,v} & u^T Y_{\cdot,k} - v^T X_{\cdot,k} \\ \text{subject to} & X\lambda + s = X_{\cdot,k} & \text{subject to} & u^T Y - v^T X \leq 0 \\ & Y\lambda - t = Y_{\cdot,k} & & u, v \geq e \\ & \lambda, s, t \geq 0 & & \end{array} \tag{2.16}$$

Note that in the primal model, the constraints correspond to:

$$X\lambda = X_{\cdot,k} - s \quad \text{and} \quad Y\lambda = Y_{\cdot,k} + t.$$

Since we are seeking to maximize s and t (by minimizing their opposites), we are actually seeking to minimize the input and maximize the output of the composite DMU — both at the same time. In the dual model, we are manipulating the weights directly to maximize the difference between the weighted output and the weighted input.

To model the VRS problem, we restrict the search for the composite DMU to the convex hull of the current DMUs by the same restriction on λ as given

in the BCC models:

$$
\begin{array}{llll}
\text{Additive Primal VRS} & & \text{Additive Dual VRS} & \\
\min\limits_{\lambda,s,t} & -(e^T s + e^T t) & \max\limits_{u,v,\mu} & u^T Y_{\cdot,k} - v^T X_{\cdot,k} + \mu \\
\text{subject to} & X\lambda + s = X_{\cdot,k} & \text{subject to} & u^T Y - v^T X + \mu e \leq 0 \\
& Y\lambda - t = Y_{\cdot,k} & & u, v \geq e \\
& e^T \lambda = 1 & & \mu \text{ free} \\
& \lambda, s, t \geq 0 & &
\end{array}
\tag{2.17}
$$

DMUs analyzed under the additive models are considered efficient when their objective values are zero — when the slack between the composite DMU and the current DMU is zero. Notice that there is no efficiency measure present in additive models; instead we look directly at *individual* inputs and outputs (through the slack variables). Thus, we have no distinction between weak and strong efficiencies, and so have no need for a second phase.

Variations and Extensions

Besides the three basic models, variations and extensions have been developed to deal with a variety of applications [24]. To show the variety of DEA of models and thus the variety of models that we must be able to consider under the slice interface, we describe some of the more common ones here.

Super-Efficiency Under the basic DEA models, inefficient DMUs can be ranked by their varying efficiency scores, but efficient DMUs cannot because they are all assigned efficiency scores of 1. To overcome this difficulty, Andersen and Petersen [4] proposed a variation on the primal input BCC model from (2.14) in which the DMU currently under evaluation is excluded from consideration in any composite DMUs. Mathematically, this corresponds to fixing $\lambda_k = 0$ in the primal model; in the dual model, this corresponds to eliminating

$$u^T Y_{\cdot,k} \leq v^T X_{\cdot,k}$$

from the constraint set. When the DMU is inefficient, this has no affect on the efficiency score; however, when the DMU is efficient, this allows for efficiency scores greater than 1. Because of this, these models are often referred to as the super-efficient models.

Non-discretionary Variables Non-discretionary variables are variables which affect output but which the DMUs have no control over [28]. Examples of such variables include weather or store location. To deal with these types of variables, Banker and Morey [7] suggested a primal input BCC model in which non-discretionary input variables are not scaled by θ and any slack

from these variables is not considered in the strong efficiency calculation:

BCC Strong Efficiency Model with Non-discretionary Variables

$$\begin{aligned} \min_{\theta,\lambda,s_D,s_N,t} \quad & \theta - \epsilon(e^T s_D + e^T t) \\ \text{subject to} \quad & \theta X_{D,k} - s_D = X_{D,\cdot}\lambda \\ & X_{N,k} - s_N = X_{N,\cdot}\lambda \\ & Y\lambda - t = Y_{\cdot,k} \\ & e^T\lambda = 1 \\ & \lambda, s_D, s_N, t \geq 0 \end{aligned} \tag{2.18}$$

where D is the set of discretionary inputs and N is the set of non-discretionary inputs. (For non-discretionary outputs, the corresponding primal output model is used.) As a result, composite DMUs are allowed to have at most the same amount of the non-discretionary inputs as the DMU under consideration. This ensures that we find composite DMUs that are no better in uncontrollable conditions. The same approach can be used for CCR models; additive models can also be extended [24, p. 51].

Categorical Variables Efficiency scores under DEA are determined by looking for composite DMUs that perform better than the current DMU under evaluation. These composite DMUs are built by taking conical or convex combinations of the actual DMUs. This works well when the DMUs

we are evaluating are in the same operational category; however, for DMUs in different categories, this may create unfair comparisons. For example, we may obtain poor results comparing banks with drive-in capabilities to banks without drive-in capabilities. To deal with situations like these, Banker and Morey [8] developed the idea of categorical variables in DEA. Borrowing notation from [2], we consider a categorical input which can take on one of the values $\{1, 2, \ldots, C\}$, where 1 is the most favorable and C is the least favorable. This categorical input then divides the set of DMUs into C disjoint categories: $D_1, D_2, \ldots, D_C$; in addition, these categories are ordered so that D_i is in a more favorable position than D_{i+1}. Assuming that the DMUs' categories cannot be changed (non-controllable categories), we only want to perform the efficiency analysis using DMUs in the same or in a less favorable category. This can be done by excluding DMUs in more favorable categories from consideration. In the following BCC model, we assume that DMU k is

in category ℓ:

BCC Model with (Non-controllable) Categorical Variables

$$\begin{aligned}
\min_{\theta,\lambda} \quad & \theta \\
\text{subject to} \quad & \theta X_{i,k} \geq \sum_{j \in D_\ell \cup D_{\ell+1} \cup \ldots \cup D_C} X_{i,j}\lambda_j \quad \text{for each input } i \\
& Y_{r,k} \leq \sum_{j \in D_\ell \cup D_{\ell+1} \cup \ldots \cup D_C} Y_{r,j}\lambda_j \quad \text{for each output } r \\
& e^T\lambda = 1 \\
& \lambda \geq 0
\end{aligned} \tag{2.19}$$

If there is no hierarchy to the categories, then we exclude DMUs in all categories except for category ℓ. For (non-controllable) categorical outputs, the equivalent output model can be used. The same approach can be used for CCR and additive models.

Model (2.19) deals with cases where individual DMUs cannot change their categories. For cases where the DMUs can change categories (controllable categories), Banker and Morey [8] proposed a model involving discrete parameters and variables. Their model was corrected by Kamakura [55], whose model was further extended by Rousseau and Semple [89]. In all three versions, an output categorical variable is assumed and a modified output-oriented BCC model is used. The main idea involves including in the model additional binary parameters $W_{\ell,j}$ which define the category level ℓ

($\ell \in \{1, \ldots, C-1\}$) for DMU j. If DMU j is in category C (least favorable), then $W_{\ell,j} = 0\ \forall \ell$; if DMU j is in category $C-1$ (second to least favorable), then $W_{1,j} = 1$ and $W_{\ell,j} = 0$ for all other ℓ; other levels are defined similarly up to the most favorable (category 1), where $W_{\ell,j} = 1\ \forall \ell$. To identify the maximum possible gain in the categorical variable, we present the following model (suggested in [89]) for DMU k:

BCC Model with Controllable Categorical Variables

$$
\begin{aligned}
\min_{\lambda,t} \quad & e^T t \\
\text{subject to} \quad & X_{\cdot,k} \geq X\lambda \\
& Y\lambda \geq Y_{\cdot,k} \\
& W_{\ell,\cdot}\lambda - t_\ell = W_{\ell,k} \quad \text{for } \ell = 1, \ldots, C-1 \\
& e^T\lambda = 1 \\
& \lambda \geq 0 \\
& t_\ell \text{ binary}
\end{aligned}
\tag{2.20}
$$

As the objective function only involves the slacks from the categorical output, model (2.20) can only identify possible improvements in that variable. To include other (continuous) input-output variables in the analysis, a two-phase approach similar to the one used for strong efficiency can be used [89]: in phase 1, deal with continuous-variable inefficiencies (using model (2.15) for

example); in phase 2, deal with categorical inefficiencies (using model (2.20) or a variant). Rousseau and Semple [89] also note that model (2.20) be formulated and solved as multiple *linear* programs instead of a single *mixed integer* program.

Model-Specific Constraints Many model extensions come from the desire to include other relevant information in the model through the use of additional constraints or restrictions [85]. This may be done to make the results consistent with "prior knowledge or accepted views" or to incorporate "value judgements" in the analysis [3]. For example, performing the basic CCR analysis on the data from the Program Follow Through application [21] (as suggested in [3]), we find some schools (such as school 15) are rated as efficient with output weights of zero for the reading and math test scores. These results imply that only the self esteem measurements, and not the test scores, were used to determine efficiency. To force consideration of the test scores, we can set positive lower bounds on their weights. Besides such direct weight restrictions, some applications may require "virtual" (indirect) weight restrictions or other constraints, which may also be DMU-specific.

General DEA Models as Slice Models

Like the CCR models, these general DEA models meet the definition of slice models. In all cases, the model structure remains constant and only the data

change between solves. This has to do with the way in which the analysis is done: one particular DMU is compared to the entire set of DMUs. When we change the DMU under consideration, the comparison data must be changed.

Note that specialized DEA solvers, which take advantage of the common structure between programs, do exist (for example, see [90]). However, these solvers do not handle more general DEA problems; typically, they solve problems (2.2) and (2.3), and a few common variations. This works well when the problem to be solved is one of the assumed forms, but when the problem is of a nonstandard form, another solver must written or a general solver must be used. As can be seen from this section, nonstandard DEA problems are quite common.

Further, we may want to employ alternate methods to measure a DMU's efficiency. For example, rather than looking at ratios of weighted output to weighted input, we could directly measure how close each DMU is to Farrell's efficiency frontier. This is addressed in [75], where Mangasarian considers the distance between a point in a polyhedral set (the DMU input-output set) and the boundary of the set (the efficiency frontier). In [75], a general polytope is considered:

$$S = \{y | y = Bz, z \geq 0, e^T z = 1\}$$

where B is a matrix of real values and e is a vector of ones. For the DEA version, B is an $n \times (|i| + |o|)$ matrix containing X and Y. Then, the distance between a point $s \in S$ and a projection $p(s)$ on the boundary is given by [75,

Theorem 2.2]:

$$\begin{aligned} \|s - p(s)\| = \quad & \min_{y,\zeta} && s^T y + \zeta \\ & \text{subject to} && B^T y + e\zeta \geq 0 \\ & && \|y\|' = 1 \end{aligned} \tag{2.21}$$

where $\|\cdot\|$ is a general norm and $\|\cdot\|'$ is the corresponding dual norm. As is noted in [75], this program is NP-hard for the 2-norm and the ∞-norm. We can solve this program though, for the 1-norm by solving $2n$ linear programs:

$$\|s - p(s)\|_1 = \min_{i=1,\ldots,n,\sigma=\pm 1} P_{i,\sigma}$$

where

$$\begin{aligned} P_{i,\sigma} = \quad & \min_{y,\zeta} && s^T y + \zeta \\ & \text{subject to} && B^T y + e\zeta \geq 0 \\ & && -e \leq y \leq e \\ & && y_i = \sigma \end{aligned} \tag{2.22}$$

Note that the programs (2.22) are also slice models, where only the constraint

$$y_i = \sigma$$

changes.

These examples show that the ability to define the model is important for DEA and related applications, suggesting the use of a more general modeling

language. In addition to allowing the user to solve nonstandard DEA models, modeling languages also give the user the ability to modify their model easily, and define new models as desired.

2.2 The Slice Interface

Solving slice models involves looping over the data slices, including specific slices in the model. Even if there are no or very little core data, the structure of each program is the same and so can be held constant. Often with general solvers, no mechanism is available to store the structure and core data; each program must be generated and solved from scratch, ignoring completely the common ties that the programs have with each other. For example, the DEA implementations [40, 85] for the GAMS modeling language make use of the GAMS loop structure within the expression of the model. This loop structure defines the particular programs which are passed to the general solver and includes the solver call. As a result, multiple calls are made to the general solver, each accompanied by a new model generation. Some optimal basis information may be passed on (depending upon option settings), but the structure and data are entirely regenerated, resulting in huge solution times for models with many DMUs.

As an alternative, we suggest that similarities between the programs be

considered. To do this, we have built an interface between the GAMS modeling language and the CPLEX callable library. This slice interface uses the GAMS dictionary to identify the changes between programs and the CPLEX problem modification routines to make the changes — all automatically. In this way, we are able to reuse the common structure and core data without having to regenerate them for each program.

2.2.1 Implementation

Unlike in the general slice models of [40, 85], *all* of the information for all of the programs is passed into the interface at once. The interface then separates this information into program structure and core rows ($\mathcal{X}$), and program-specific rows (A^k and b^k), based upon the modeler's instructions. Once the interface has labeled the constraints and received the data from GAMS, the core rows and structure are read into a CPLEX problem object. At this stage, we solve the slice model. This is done in a loop (replacing the loop previously written in GAMS) over the slice set. At each iteration, specific rows are added to the core program object. The resulting program is then solved and its results are printed to solution files. Next, the specific program is returned to its core state while some solution information (basis information for linear programs and starting point information for mixed integer programs) is passed on to the next solve. Once the last solve is done, we return the last solution to GAMS. In this way, the slice interface

enables us to perform the multiple solves needed by slice models, while at the same time keeping program structure constant and using advance starting information.

To instruct the interface on which constraints are core constraints and which are program-specific, the modeler needs to do very little. First the slice set must be identified. This is done by associating a special name (`slice`) with the slice set through an alias command. Then, the constraints which contain slice data must be identified. This is done by *declaring* these constraints over the special alias name; these constraints can then be defined over any subset of the slice set. Core constraints are declared and defined normally. Note that we do not have to worry about changing objective functions: in GAMS the objective function is given as an objective variable — the data associated with the objective function are actually stored in the constraint matrix.

Figure 1 compares the general GAMS implementation to the GAMS slice implementation for the dual DEA model (2.2), and shows that very little must be changed. First, the set of DMUs (`n`) which are being evaluated are identified as the slice set. Then, the slice constraints (`objfcn` and `denom`) are declared over the special alias name `slice`.

Although a direct translation from the general GAMS implementation is easy to produce for the GAMS slice implementation, it is not always the

```
sets  n    'DMUs',
      k(n) 'selected unit',
      i    'inputs',
      o    'outputs';



equations
objfcn(n) 'eff defn',
denom(n)  'wted input',
lime(n)   'out/in<1';

objfcn(k)..
sum(o, u(o)*Y(o,k))
=e= eff;

denom(k)..
sum(i, v(i)*X(i,k))
=e= 1;

lime(n)..
sum(o, u(o)*Y(o,n))
=l= sum(i, v(i)*X(i,n));



set nloop(n) 'DMUs to use';
* run over all DMUs
nloop(n) = yes;

k(n) = no;
loop(nloop,
  k(nloop) = yes;
  solve dea using lp max eff;
  k(nloop) = no;
);
```

(a)

```
sets  n    'DMUs',
      k(n) 'selected units',
      i    'inputs',
      o    'outputs';
alias(n,slice);



equations
objfcn(slice) 'eff defn',
denom(slice)  'wted input',
lime(n)       'out/in<1';

objfcn(k)..
sum(o, u(o)*Y(o,k))
=e= eff;

denom(k)..
sum(i, v(i)*X(i,k))
=e= 1;

lime(n)..
sum(o, u(o)*Y(o,n))
=l= sum(i, v(i)*X(i,n));



* run over all DMUs
k(n) = yes;




solve dea using lp max eff;
```

(b)

Figure 1: DEA model implementation for (2.2) using (a) the general GAMS interface and (b) the GAMS slice interface.

best formulation to use. Under our approach, the entire problem is generated and passed to the slice interface initially. This works fine for models like model (2.2) because the extra constraints that we must generate are few in comparison to the whole model. However, in other models like model (2.3), almost all of the constraints change and so many more constraints are passed initially to the solver. This results in very high problem generation times, which can destroy any time-savings the slice interface achieves during the actual solves. Dealing with this issue involves adjusting the model to fit the way the solver works: models like model (2.3) change by column slices (variables and right-hand-side values); the slice interface, on the other hand, makes use of row slices (constraints and objective function coefficients). In modeling using column slices, many constraints change overall, but the changes within individual constraints are minor. As a result, we end up regenerating pieces of the model — exactly one of the problems that we hoped to eliminate by using the slice interface. However, by storing the pieces of these constraints that remain constant in auxiliary variables, as presented in equations (2.12) and (2.13), we are able to eliminate much of this regeneration: we still must generate many more constraints, but these constraints now have few entries in them. Although this increases the number of variables in the model, these extra variables can be eliminated by the solver during presolve, once they are no longer useful. Figure 2 compares the direct GAMS translation of model (2.3) to the improved model formulation with auxiliary variables (`fx`

and `fy`). Using auxiliary variables significantly reduces problem generation time, since the data X and Y are only generated once.

2.2.2 Technical Issues

To implement the slice procedure, we first must be able to identify the slices. This is done using the GAMS dictionary file. Under GAMS, additional information about the model is available through the dictionary file. Contained within the dictionary is the unique element list, which lists and indexes the unique element names that have been used in the model. Inside this list, alias names are listed separately, but linked to the index for their corresponding element. In a similar manner, equation names are linked to the elements that they are declared over. So, program-specific constraints can be identified by searching equation declarations for the slice alias. Because constraints are only generated for sets which they are declared over, we are able to determine which DMUs (if any) are to be skipped in the analysis simply by examining which constraints are actually generated.

After implementing our slice interface, we encountered scaling problems within our solution loop. Under default CPLEX scaling, the entire program is scaled when it is initially read in. However, it is not fully rescaled when the program is changed. As a result, we encountered numerical errors in some solutions due to poor scaling. To correct this, we unscale the problem prior to changing it. This enables us to return the problem to its true core state

```
equations
di(slice,i) 'input constr',
do(slice,o) 'output constr';



di(k,i)..
theta*X(i,k) =g=
sum(n, X(i,n)*lambda(n));

do(k,o)..
sum(n, Y(o,n)*lambda(n))
=g= Y(o,k);
```

(a)

```
equations
di(slice,i) 'input constr',
do(slice,o) 'output constr',
dfx(i)      'aux input var',
dfy(o)      'aux output var';

di(k,i)..
theta*X(i,k) =g= fx(i);


do(k,i)..
fy(o) =g= Y(o,k);


dfx(i)..
fx(i) =e=
sum(n, X(i,n)*lambda(n));

dfy(i)..
fy(o) =e=
sum(n, Y(o,n)*lambda(n));
```

(b)

Figure 2: Column-based slice problem (2.3) under (a) direct translation into GAMS and under (b) auxiliary variables.

and guarantees that when scaling is done (for instance, during the solve), it is done to the problem being solved and not just pieces of the problem.

Besides these implementation issues, we must also deal with the issue of solution availability. Under the GAMS loop implementation, solution information for each program is accessible from within GAMS because each program's solution is returned to GAMS. Under the GAMS slice implementation, only the solution of the last slice problem is returned to GAMS. Initially, solution information for all of the programs was written to a solution file as GAMS parameter values indexed by slice [38]. However, GAMS Development Corporation created new procedures, GDXread and GDXwrite, which allow for run-time reading and writing of files [81]. Using these GDX procedures, we are able to write out solution files which can be read in immediately after the solve completes. This is advantageous for situations in which additional computation is necessary. Using the GDX files, such additional computation can be done from within the same model file and within the same GAMS call — no additional calls are needed. For example, within the same run, we can combine the results from multiple DEA solves into confidence intervals using the process described in Section 2.3. For the cross-validation problems discussed in Section 2.4, we are able to perform training and testing all at once.

Under defaults, the model status, solver status and objective values are written to GDX files. Other variable and constraint values can also be written

to GDX files (one file for each value type) using GAMS options. Prior to the solve, the interface informs the user of the GDX file(s) being used. Under GDX files, users can perform additional analysis on their results immediately in the same GAMS file.

Although the GAMS modeling language allows us to define many different types of mathematical programming models, the slice interface is limited by its solver, CPLEX. As a solver, CPLEX can solve linear, mixed-integer, and quadratic (through the use of its barrier method) programs. As a result, the slice interface is limited to linear, mixed-integer and simple quadratic (where the Q matrix is constant) models. At this time, there is not an equivalent interface for nonlinear models.

One thing to note is that since the slice interface is a general solver for slice models, it does not incorporate specific DEA information into the solution process. Under the DEA analysis, when a DMU is found to be inefficient, the reference DMUs (those making up the composite DMU in model (2.3)) are efficient, and so need not be evaluated later. Specialized DEA solvers automatically assign these DMU's an objective value 1 and do not evaluate them. The slice interface does not do this. Although this will increase the solution time (performing a solve for a problem whose solution is already known), this allows the interface to be used for general slice models, whether they are variations on DEA problems or another type of slice model altogether.

2.2.3 Solution Options

Under GAMS, solver behavior can be modified through the use of an options file. Many options are available for the slice interface. One is the choice of solution algorithm. For linear programs, the slice interface initially chooses the dual simplex method for minimization models and the primal simplex method for maximization models. These defaults were chosen based upon the performance of our interface under test problems, but can be changed in the options file. Mixed integer problems automatically use the CPLEX MIP solver. Simple quadratic programs use the CPLEX barrier solver.

Advanced starting information is also important to the slice interface. When an advanced starting basis is used, CPLEX (version 6.6) skips the presolve stage. So, whether an advanced basis is used or not greatly affects how the individual programs are solved. Our tests on general linear DEA problems indicate that using advanced basis information (and skipping the presolve stage) in subsequent solves can greatly improve solution time. Because changing a program destroys advanced starting information, we explicitly copy the advanced basis from one program to the next in the interface code. The copy, though, is ignored if use of the advance basis is turned off in the options file.

A complete listing of options for the slice interface is available from [41].

2.2.4 DEA Applications

To test the slice interface, we consider two DEA applications. For all examples, we compare the original GAMS formulations under CPLEX 6.6 to the corresponding GAMS slice formulations under the slice interface. All presented results were run on a Sun Ultra 10 440 Mhz workstation running Unix (Solaris 8).

The first DEA application measures the efficiency of 104 German hospitals [60]. In this application, the inputs included the number of beds and the annual cost of care, while the outputs included the number of cases per hospital department (for the 1992 data used in [60], 18 hospital departments were considered). To determine efficiency for each hospital, we used model (2.3), which resulted in 49 efficient hospitals and 2 very inefficient (efficiency scores less than 10%) hospitals. The time comparison between the GAMS model under CPLEX and the GAMS model under the slice interface are given in the first row of Table 1. For this example, the time for the original solve is insignificant (around seven seconds), but the huge time savings that the slice interface achieves even at this level suggest that it is beneficial to much larger problems.

Another application, measuring the capacity of a fishing fleet, truly shows the power of the slice interface. The fishing vessels in the fleet make up the set of DMUs, and the amounts of different fish caught make up the outputs.

The inputs come from both discretionary variables (like crew size) and non-discretionary variables (like the physical characteristics of the vessels). To do the analysis, capacity was defined as the maximum amount that can be produced provided the availability of variable (discretionary) inputs is not restricted [52, p. 52]. This definition was converted into the following BCC primal output-oriented DEA model involving non-discretionary inputs (a variation on model (2.18)) [107]:

$$\begin{aligned} \max_{\phi,\lambda,\gamma} \quad & \phi \\ \text{subject to} \quad & \widetilde{Y}\lambda \geq \phi \widetilde{Y}_{\cdot,k} \\ & X_{N,\cdot}\lambda \leq X_{N,k} \\ & X_{D,\cdot}\lambda = \gamma_{D,k} \cdot X_{D,k} \\ & e^T\lambda = 1 \\ & \gamma, \lambda \geq 0 \end{aligned} \tag{2.23}$$

In this model, we are looking for a composite DMU that produces more averaged yearly output (represented by $\widetilde{Y}$) than our current DMU. Here, ϕ represents the factor that gives how much more output the composite DMU produces, and so represents a capacity measure for the current DMU. The constraints for the non-discretionary inputs are the natural output-oriented constraints, but the constraints for the discretionary inputs differ. Using the definition given above for capacity, we do not want to restrict discretionary

Application Problem	General Solver (CPLEX)	Slice Interface
Hospital Problem (104 DMUs)	7.02 sec	0.51 sec
Small Problem (60 DMUs)	4.63 sec	0.43 sec
Large Problem (4888 DMUs)	16 hours	0.2 hours

Table 1: Time comparisons between the general GAMS implementations and the GAMS slice implementations.

inputs. So we include an additional variable $\gamma_{D,k}$ for each discretionary input D and each DMU k, which lets us adjust the discretionary inputs as needed to meet those of the composite DMU. Finally, we use the variable returns to scale constraint, $e^T\lambda = 1$.

We consider two different problems provided by Walden [107]: the small problem involves 60 DMUs, 3 non-discretionary inputs, 1 discretionary input, and 6 outputs; the large problem involves 4888 DMUs, 4 non-discretionary inputs, 1 discretionary input, and 4 outputs. Results comparing the GAMS model under CPLEX and the GAMS model under the slice interface are given in the second and third rows of Table 1. In the small problem we see some time improvement, just as we did for the hospital application. But, in the large problem, we see a significant improvement in time: 16 hours versus approximately 12 minutes. This result especially shows the power of the slice interface for DEA models: we can solve *general* DEA models with many DMUs much more efficiently.

2.3 Beyond DEA: Sensitivity Analysis

Because of the time improvements over general GAMS implementations, slice model solutions can be extended through the application of other procedures to include additional information like confidence intervals. In finding efficiency scores under DEA, we use the provided DMU data to determine feasible input-output points (both actual DMU points and composite DMU points). DEA efficiency scores are then determined based upon these feasible points. Thus, changes in the DMU data result in changes to the feasible points and to the efficiency scores. To determine the sensitivity of DEA scores to changes in the data, a variety of techniques have been proposed, from using aspects of traditional linear programming sensitivity analysis to statistical approaches.

2.3.1 Types of Sensitivity Analysis

Sensitivity analysis in linear programming involves investigating the effects that changes in the data have on a problem. Techniques for performing such analysis tend to assume that either primal or dual feasibility is maintained [27, 83]. But, for DEA models, such assumptions do not hold because changing DMU data can easily change basic variable coefficients (particularly when the current DMU is efficient). Under such cases, Chvátal [27] and Murty [83] recommended introducing new variables to handle the new

data. Although practical for considering the effects of changing the data associated with a single DMU, such approaches are impractical for considering changes in all of the data. Further, because DEA models tend to have multiple optimal solutions (for example, the strong efficiency analysis looks for other optimal solutions) and tend to be highly degenerate (for example, an efficient DMU analyzed under the primal CCR model has only two nonzero basic variables, λ_k and θ), problems may arise in the analysis [22].

Variations on linear programming approaches have also been proposed. Charnes and Neralic [23] developed sufficient conditions for testing whether a perturbation added to an efficient DMU (as determined by the additive model) preserves efficiency; this is done by maintaining feasibility through updates to the inverse of the optimal basis matrix. However, this tends to result in overly restrictive sufficient conditions [22]; further, it only examines data changes in the DMU currently under consideration. Thompson et al. [101] looked at the effects of changing all of the data: they use the strong complementary slackness condition to analyze the CCR model by changing efficient and inefficient data simultaneously in opposite directions. Their results though, tend to be inexact and depend upon the complementary slackness condition employed [93].

Other sensitivity techniques involve using modified DEA models. In [22], Charnes et al. built modified models which find DMU k's radius of stability R_p, the largest number such that perturbations to DMU k with p-norm

strictly less than R_p do not change DMU k's state (efficient or inefficient). The larger the radius of stability, the less sensitive DMU k is to data changes. Charnes et al. [22] proved that R_∞ for an (additive) efficient DMU k is given by the solution to the following modified additive DEA model:

$$\begin{aligned} \min_{\lambda,s,t,\beta} \quad & \beta \\ \text{subject to} \quad & X\lambda + s = X_{\cdot,k} + \beta e \\ & Y\lambda - t = Y_{\cdot,k} - \beta e \\ & \lambda_k = 0 \\ & \lambda, s, t, \beta \geq 0 \end{aligned} \tag{2.24}$$

Here, DMU k has been removed from the composite DMU set by fixing $\lambda_k = 0$. Essentially, this model seeks to determine how much DMU k can be worsened (by increasing its inputs and decreasing its outputs uniformly by β) before it becomes inefficient. A similar program is used for inefficient units, where β is used instead to decrease the inputs and increase the outputs. In the inefficient model, the DMU does not need to be omitted from the composite DMU set [22].

The models proposed in [22] only allow the DMU currently under consideration (DMU k) to change. Seiford and Zhu [93] proposed variations which allow all of the data to change. In order to do the analysis, they assumed a worst-case scenario by assuming that an efficient DMU k deteriorates while

the remaining DMUs improve. For CCR and BCC models, they considered percentage changes in a subset of the inputs ($i \in I$) and a subset of the outputs ($r \in O$):

$$\hat{X}_{i,k} = \begin{cases} \delta X_{i,k} = X_{i,k} + (\delta - 1)X_{i,k} & \text{for } \delta \geq 1, i \in I \\ X_{i,k} & \text{for } i \notin I \end{cases}$$

$$\hat{Y}_{r,k} = \begin{cases} \tau Y_{r,k} = Y_{r,k} - (1 - \tau)Y_{r,k} & \text{for } 0 < \tau \leq 1, r \in O \\ Y_{r,k} & \text{for } r \notin O \end{cases}$$

(so that the efficient DMU k deteriorates) and for $j \neq k$,

$$\hat{X}_{i,j} = \begin{cases} X_{i,j}/\delta = X_{i,j} - \frac{\delta - 1}{\delta} X_{i,j} & \text{for } \delta \geq 1, i \in I \\ X_{i,j} & \text{for } i \notin I \end{cases}$$

$$\hat{Y}_{r,j} = \begin{cases} Y_{r,j}/\tau = Y_{r,j} - \frac{1 - \tau}{\tau} Y_{r,j} & \text{for } 0 < \tau \leq 1, r \in O \\ Y_{r,j} & \text{for } r \notin O \end{cases}$$

(so that DMU j improves). Values for δ and τ which keep DMU k efficient under simultaneous changes in both inputs and outputs can be found by

solving the following model from [93], which is closely related to model (2.24):

$$\begin{aligned}
\Gamma^* = \min_{\lambda,s,t,\Gamma} \quad & \Gamma \\
\text{subject to} \quad & X_{i,\cdot}\lambda \leq (1+\Gamma)X_{i,k} \quad \text{for } i \in I \\
& X_{i,\cdot}\lambda \leq X_{i,k} \quad \text{for } i \notin I \\
& Y_{r,\cdot}\lambda \geq (1-\Gamma)Y_{r,k} \quad \text{for } r \in O \\
& Y_{r,\cdot}\lambda \geq Y_{r,k} \quad \text{for } r \notin O \\
& \lambda_k = 0 \\
& \lambda \geq 0
\end{aligned}$$

If

$$1 \leq \delta \leq \sqrt{1+\Gamma^*}$$

and

$$\sqrt{1-\Gamma^*} \leq \tau \leq 1,$$

then DMU k remains efficient [93]. A similar model is also considered for absolute changes in the data of BCC models.

Rather than looking at all of the data for all DMUs, Kuntz and Scholtes [60] focused on specific, problem-dependent data elements. They considered efficiency status *robustness*, defined to be the minimal perturbation of chosen data elements that is necessary to change the efficiency status of a DMU. They illustrated their technique on the set of 104 German hospitals from

Section 2.2.4. Once DEA efficiency scores were obtained, inefficient hospitals were analyzed with respect to the specific input, the number of beds. This was done by finding the minimal number of beds which must be redistributed from an inefficient hospital to the efficient hospitals in order to make the inefficient hospital efficient. For an inefficient hospital k, this is achieved by solving a modified super-efficient model (see Section 2.1.4):

$$\begin{aligned}
\min_{u,v,s} \quad & e^T s \\
\text{subject to} \quad & u^T Y_{\cdot,e} \leq v_1 X_{1,e} + 1 X_{2,e} + s_e \qquad \text{for } e \in E \\
& u^T Y_{\cdot,j} \leq v_1 X_{1,j} + 1 X_{2,j} \qquad \text{for } j \notin E, j \neq k \\
& u^T Y_{\cdot,k} \geq v_1 X_{1,k} + 1 X_{2,k} - e^T s \\
& u, v, s \geq 0
\end{aligned}$$

where E is the set of efficient DMUs. Here, input 1 refers to the annual cost of care and input 2 refers to the number of beds. We essentially fix the input weight on the number of beds (the data element under consideration) to be 1, so that we deal with the *full* bed count. Then, we find the minimum number of redistributed beds, taken from DMU k and given to the efficient DMUs, such that

$$u^T Y_{\cdot,k} \geq v_1 X_{1,k} + 1 X_{2,k} - e^T s \quad \Rightarrow \quad \frac{u^T Y_{\cdot,k}}{v_1 X_{1,k} + 1 X_{2,k} - e^T s} \geq 1,$$

implying that DMU k is at least efficient (and may be super-efficient, as its

efficiency score is not restricted to be less than or equal to 1 in the second constraint).

The approaches discussed above all use only the DMU data to evaluate the sensitivity of the DEA efficiency scores in some way. An alternative is to obtain additional feasible input-output points, against which we can measure the efficiency scores for the original DMUs. Simar and Wilson [97, 98, 99] presented two techniques for sampling from the DMU data under a smoothed bootstrap and evaluating efficiency based on these samples. Below, we discuss their homogeneity approach from [97], focusing on the input-oriented CCR models (2.2) and (2.3).

2.3.2 Simar and Wilson's Sensitivity Analysis

Typically, DEA efficiency scores are taken to be measures of a DMU's true efficiency. Under Simar and Wilson's approach, these scores are considered to be only estimates.

True Efficiency Scores

Given a vector of inputs x and a vector of outputs y, Simar and Wilson [97] define the *production possibility set* Ψ to be the set of all physically attainable input-output points (x, y):

$$\Psi = \{(x, y) | \text{ outputs } y \text{ can be produced from inputs } x\}. \tag{2.25}$$

Note that by definition the DMU data $(X, Y) \in \Psi$. To be consistent with the DEA approach, Ψ is defined to be a convex cone:

Assumption 2.6 *Composite input-output points are physically attainable:*

$$(x_{\cdot,k}, y_{\cdot,k}) \in \Psi \quad \Rightarrow \quad \left(\sum_k x_{\cdot,k}\lambda_k, \sum_k y_{\cdot,k}\lambda_k \right) \in \Psi \text{ for } \lambda_k \geq 0.$$

(For variable returns to scale, which is used in [97], the additional condition $\sum \lambda_k = 1$ is added so that Ψ is a convex set.) This assumption comes from the way in which DMUs are evaluated under DEA. Under the CCR version of DEA, we compare the current input-output point to a composite point built by taking conical combinations of the known points. If the composite point were not physically attainable, our comparison would be invalid.

Ψ can be also described by input and output requirements. Borrowing notation from [95], for every input x, the set of outputs that x can physically produce is given by the *production correspondence*:

$$P(x) = \{y | (x, y) \in \Psi\}. \tag{2.26}$$

Likewise, for every output y, the set of inputs which can physically produce y is given by the *inverse production correspondence*:

$$L(y) = \{x | (x, y) \in \Psi\}. \tag{2.27}$$

Relationships between the different correspondences are given by the following standard production assumption from [95]:

Assumption 2.7 *(Disposability) We can dispose of excess inputs or outputs without penalty:*

$$y_{\cdot,1} \leq y_{\cdot,2} \quad \Rightarrow \quad L(y_{\cdot,2}) \subseteq L(y_{\cdot,1})$$

(inputs used to produce more output can also be used to produce less) and

$$x_{\cdot,1} \leq x_{\cdot,2} \quad \Rightarrow \quad P(x_{\cdot,1}) \subseteq P(x_{\cdot,2})$$

(outputs produced from less input can also be produced from more).

Associated with each input-output point $(x_{\cdot,k}, y_{\cdot,k})$ is a true input efficiency measurement θ_k, defined by [97]:

$$\theta_k = \min\{\theta | \theta x_{\cdot,k} \in L(y_{\cdot,k})\}. \tag{2.28}$$

(Output efficiency measurements are defined similarly over $P(x_{\cdot,k})$.) θ_k represents the feasible proportionate reduction in inputs that is needed to make the input-output point $(x_{\cdot,k}, y_{\cdot,k})$ efficient. If $\theta_k = 1$, then the point $(x_{\cdot,k}, y_{\cdot,k})$ is considered to be efficient; if $\theta_k < 1$, then the point is inefficient. The corresponding efficient input level is given by [99]

$$x^{\partial}(y_{\cdot,k}) = \theta_k x_{\cdot,k}. \tag{2.29}$$

The input efficient pair $(x^{\partial}(y_{\cdot,k}), y_{\cdot,k})$ always lies on the *true efficiency frontier*, which envelops the production possibility set. (This is the frontier which Farrell's approach [34] and DEA attempt to estimate.) Because efficient points have $\theta_k = 1$, these points themselves lie on the efficiency frontier.

Notice that the minimization is done over *all* physically attainable sets (that is, Ψ), and so θ_k and $x^{\partial}(y_{\cdot,k})$ do not depend upon the DMU data (X, Y) and do not change as the DMU data changes.

DEA Efficiency Scores

Under DEA we have only a subset of Ψ, namely (X, Y), corresponding to the data for the n DMUs under consideration. The remainder of Ψ is unknown, and thus, so are the correspondences P and L, and the true efficiency scores θ_k. In [97], Ψ is approximated using X, Y and Assumptions 2.6 and 2.7:

$$\hat{\Psi} = \{(x, y) | y \leq Y\lambda;\ x \geq X\lambda;\ \lambda \geq 0\} \tag{2.30}$$

(Here again, the additional constraint $e^T\lambda = 1$ is included in [97] for variable returns to scale.)

Theorem 2.8 $\hat{\Psi} \subseteq \Psi$.

Proof: Let $(\hat{x}, \hat{y}) \in \hat{\Psi}$. From equation (2.30) and Assumption 2.7,

$$\hat{y} \leq Y\lambda \quad \Rightarrow \quad L(Y\lambda) \subseteq L(\hat{y})$$

and

$$\hat{x} \geq X\lambda \ \Rightarrow \quad P(X\lambda) \subseteq P(\hat{x}).$$

So, under Assumption 2.6 and these relationships,

$$(X\lambda, Y\lambda) \in \Psi \quad \Rightarrow \quad (X\lambda, \hat{y}) \in \Psi \quad \Rightarrow \quad (\hat{x}, \hat{y}) \in \Psi. \ \square$$

Approximations for P and L can be obtained using $\hat{\Psi}$:

$$\hat{P}(x) = \{y|(x,y) \in \hat{\Psi}\} \tag{2.31}$$

and

$$\hat{L}(y) = \{x|(x,y) \in \hat{\Psi}\}. \tag{2.32}$$

Input efficiency can be approximated for each $(x_{\cdot,k}, y_{\cdot,k}) \in \hat{\Psi}$ by

$$\hat{\theta}_k = \min\{\theta|\theta x_{\cdot,k} \in \hat{L}(y_{\cdot,k})\}.$$

In particular, for a DMU point $(X_{\cdot,k}, Y_{\cdot,k})$,

$$\hat{\theta}_k = \min\{\theta|Y_{\cdot,k} \leq Y\lambda; \theta X_{\cdot,k} \geq X\lambda; \lambda \geq 0\}, \tag{2.33}$$

the original (CCR) DEA input efficiency score. The corresponding efficient input level for $(x_{\cdot,k}, y_{\cdot,k}) \in \hat{\Psi}$ is then approximated by:

$$\hat{x}^{\partial}(y_{\cdot,k}) = \hat{\theta}_k x_{\cdot,k}. \tag{2.34}$$

Notice that we can recover $x_{\cdot,k}$ from $\hat{x}^{\partial}(y_{\cdot,k})$ by

$$x_{\cdot,k} = \frac{\hat{x}^{\partial}(y_{\cdot,k})}{\hat{\theta}_k}. \tag{2.35}$$

Theorem 2.9 *For* $(x_{\cdot,k}, y_{\cdot,k}) \in \hat{\Psi}$, $\hat{\theta}_k \geq \theta_k$.

Proof: From Theorem 2.8, $\hat{\Psi} \subseteq \Psi$ and so, $\hat{L}(Y_{\cdot,k}) \subseteq L(Y_{\cdot,k})$. Thus, we have $\hat{\theta}_k \geq \theta_k$. □

From equation (2.30), $\hat{\Psi}$ — and, by association, $\hat{\theta}$ and $\hat{x}^{\partial}(y)$ — are dependent upon the DMU data (X, Y). Therefore, any data variations in the DMU sample may cause changes in $\hat{\theta}$ and $\hat{x}^{\partial}(y)$. To evaluate the sensitivity of these measurements, we would like to obtain additional data points to test the DMUs against.

Obtaining Additional Data Points

Simar and Wilson [97, 98, 99] assume that some underlying data generating process $\mathcal{P}$ generates data points from Ψ. They attempt to develop a reasonable estimate, $\hat{\mathcal{P}}$, of this process in order to obtain additional data points to test against. To do this they assume that the DMU data points are random samples[1] drawn from Ψ. Then, using a smoothed, reflected bootstrap estimate for the probability density function f, they draw additional samples.

Rather than treating the DMU data (X, Y) as an i.i.d. sample obtained from (the unknown probability density function) f, Simar and Wilson take into consideration equation (2.35):

$$(X_{\cdot,i}, Y_{\cdot,i}) = \left(\frac{\hat{x}^{\partial}(Y_{\cdot,i})}{\hat{\theta}_i}, Y_{\cdot,i} \right).$$

Then $(X_{\cdot,i}, Y_{\cdot,i})$ can be thought of as being generated by (the random variables) $\hat{\theta}_i$, given $Y_{\cdot,i}$ and the efficient input measure $\hat{x}^{\partial}(Y_{\cdot,i})$ [97]. To eliminate

[1] This may be an invalid assumption: the DMU data points actually exist and so are not realizations of a random variable since there are no choices for their values. We would like to thank Professor Stephen Robinson for this observation. We would also like to note that, despite this issue, the Simar and Wilson procedure has become popular in the DEA community.

these conditions, Simar and Wilson make the "homogeneity assumption" [99]:

Assumption 2.10 $f = f(\theta|x^{\partial}, y) = f(\theta)$.

Thus, $\{\hat{\theta}_i\}$ is treated as an i.i.d. sample obtained from f. To obtain a consistent estimator, the empirical density function $\hat{f}(\{\hat{\theta}_i\})$ is smoothed:

$$\hat{f}_s(t) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{t - \theta_i}{h}\right) \tag{2.36}$$

where h is the (fixed) smoothing parameter and K is a kernel function [96]. To incorporate the bounds from Theorem 2.2 into the estimator, Simar and Wilson use the *reflection method* from [96]. The reflection method involves adding the reflections of all the points about the boundaries to the data set, and building the density estimator from the augmented data set. Including the reflected points $-\hat{\theta}_i$ and $2 - \hat{\theta}_i$ in equation (2.36) yields the density estimator:

$$\hat{f}_{sr}(t) = \frac{1}{nh} \sum_{i=1}^{n} \left[K\left(\frac{t - \hat{\theta}_i}{h}\right) + K\left(\frac{t + \hat{\theta}_i}{h}\right) + K\left(\frac{t - 2 + \hat{\theta}_i}{h}\right) \right] \tag{2.37}$$

$\hat{f}_{sr}$ is then a consistent estimator for f [92].

Simar and Wilson [97] choose $K = \phi$, the standard normal probability density function:

$$\phi(z) = \frac{1}{\sqrt{2\pi}} \exp(-z^2/2). \tag{2.38}$$

Not only is ϕ symmetric about zero, but ϕ also is nonnegative, continuous and has derivatives of all orders — properties which $\hat{f}_{sr}$ inherits [96].

Silverman [96] notes that for kernels symmetric about a boundary (like ϕ), only points near the boundary will have reflections which influence the estimator. In particular, Silverman states that if K is symmetric about zero, then we need not reflect points $\hat{\theta}_i > 4h$. In [97], Simar and Wilson consider an example in which every DMU is at least 80% efficient, and so, using $h = 0.007, 0.014, 0.028$, they drop the reflection at zero.

As noted in [96], $\hat{f}_{sr}$ does not need to be known explicitly to obtain new samples from it. Instead, Simar and Wilson [97] use the following algorithm (for completeness, we include the reflection at zero):

Algorithm 2.1 *Algorithm for Obtaining a New Sample from* $\{\hat{\theta}_i\}$

1. *Generate the* simple bootstrap sample $\beta_1^*, \ldots, \beta_n^*$ *by drawing uniformly with replacement from* $\{\hat{\theta}_1, \ldots, \hat{\theta}_n\}$. *(This step provides us with a sample from the empirical density function.)*

2. *Perturb the* β_i^* *to obtain the* bootstrap sample $\widetilde{\theta}_i^*$ *by computing:*

$$\widetilde{\theta}_i^* = \begin{cases} \beta_i^* + h\epsilon_i^* & \text{if } 0 \leq \beta_i^* + h\epsilon_i^* \leq 1 \\ -\beta_i^* - h\epsilon_i^* & \text{if } \beta_i^* + h\epsilon_i^* < 0 \\ 2 - \beta_i^* - h\epsilon_i^* & \text{if } \beta_i^* + h\epsilon_i^* > 1 \end{cases}$$

where ϵ_i^* *is a random deviate drawn from the standard normal. (In this step, we perturb the simple bootstrap sample by a random value whose probability density function is the kernel function. Notice that*

this random value is obtained from the original kernel function; thus, we may obtain values that lie outside of the boundary. If this happens, we reflect the values. This corresponds to applying the reflection method in $\hat{f}_{sr}$.)

3. *Compute the* corrected bootstrap sample*:*

$$\theta_i^* = \overline{\beta}^* + \frac{1}{\sqrt{1 + h^2/\sigma^2}}(\widetilde{\theta}_i^* - \overline{\beta}^*).$$

Here, $\overline{\beta}^*$ is the mean of the β_i^*:

$$\overline{\beta}^* = \frac{1}{n}\sum_{i=1}^{n} \beta_i^*,$$

and σ^2 is the sample variance of the $\hat{\theta}_i$:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(\hat{\theta}_i - \overline{\theta})^2$$

where $\overline{\theta}$ is the mean of the $\hat{\theta}_i$: $\overline{\theta} = \frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i$. (Making this correction ensures that the sampled values have the same mean and variance as the original values [96].)

From Algorithm 2.1, Simar and Wilson obtain a new sample $\{\theta_i^*\}$, which they use along with equation (2.35) to obtain a new data point:

$$(X_{\cdot,i}^*, Y_{\cdot,i}^*) = \left(\frac{\hat{x}^{\partial}(Y_{\cdot,i})}{\theta_i^*}, Y_{\cdot,i}\right) = \left(\frac{\hat{\theta}_i}{\theta_i^*}X_{\cdot,i}, Y_{\cdot,i}\right) \tag{2.39}$$

for $i = 1, \ldots, n$. (X^*, Y^*) is then used as data in computing new efficiency measures for each DMU. Because the efficiency measures cannot be obtained

through an explicit function, they apply the Monte Carlo Bootstrap Algorithm from [31]. This leads to the complete algorithm from [97]:

Algorithm 2.2 *The Smooth Homogeneous Bootstrap Algorithm for DEA*

1. *Compute the original efficiency measures* $\{\hat{\theta}_i \mid i = 1, \ldots, n\}$ *for the given input-output data* (X, Y).

2. *Using Algorithm 2.1, generate a new sample* $\{\theta_i^* \mid i = 1, \ldots, n\}$.

3. *Compute new data* (X^*, Y^*) *using equation (2.39).*

4. *Find the* bootstrap efficiency estimates $\{\hat{\theta}_i^* | i = 1, \ldots, n\}$ *by solving the DEA model for each DMU but using the data* (X^*, Y^*). *For example, for DMU* k *under the model (2.3), we solve*

$$\begin{aligned} \min_{\theta,\lambda} \quad & \theta \\ \text{subject to} \quad & \theta X_{\cdot,k} \geq X^* \lambda \\ & Y^* \lambda \geq Y_{\cdot,k} \\ & \lambda \geq 0 \end{aligned} \tag{2.40}$$

   to obtain $\hat{\theta}_k^*$.

5. *Repeat steps 2-4 for* $b = 1, \ldots, B$ *to obtain a set of* B *efficiency measurements* $\{\hat{\theta}_{k,b}^* \mid b = 1, \ldots, B\}$ *for each DMU* k.

In this process, although the sampling is done over the set of DMU scores, a completely new set of data is obtained at each iteration. Comparing each DMU to each new set of data allows Simar and Wilson to obtain B new efficiency measurements for each DMU k.

Under Simar and Wilson's analysis [97, 98, 99], the bootstrap efficiency scores $\hat{\theta}_k^*$ represent approximations to the $\hat{\theta}_k$, just as the DEA efficiency scores $\hat{\theta}_k$ represent approximations to the θ_k. Under this analogy, the relationship between $\hat{\theta}_k^*$ and $\hat{\theta}_k$ is equivalent to that between $\hat{\theta}_k$ and θ_k as given in Theorem 2.9. Thus:

Theorem 2.11 $\hat{\theta}_k^* \geq \hat{\theta}_k$.

Because the proof for Theorem 2.11 is rather involved, at this point we simply state the result and refer the reader to the end of this subsection (Page 75) for the details.

Confidence Intervals

Upon finishing Algorithm 2.2, Simar and Wilson have a collection of B efficiency scores for each DMU. Using these scores, they build confidence intervals for each DMU that represent the sensitivity of the efficiency scores to sampling variations of the observed frontier [97]. To find confidence intervals, they propose the modified percentile method [99], which automatically corrects for bias without the use of a bias estimator. In [99], Simar and Wilson

would like to find a_α and b_α such that

$$\text{Prob}(-b_\alpha \leq \hat{\theta}_k - \theta_k \leq -a_\alpha) = 1 - 2\alpha. \tag{2.41}$$

However, because they do not know the true efficiency scores θ_k, they use the bootstrap approximation to equation (2.41):

$$\text{Prob}(-\hat{b}_\alpha \leq \hat{\theta}^*_k - \hat{\theta}_k \leq -\hat{a}_\alpha) \approx 1 - 2\alpha. \tag{2.42}$$

$\hat{a}_\alpha$ and $\hat{b}_\alpha$ are easily found: sort the values $(\hat{\theta}^*_{k,b} - \hat{\theta}_k)$, delete $\alpha \cdot 100$ percent of the elements from both ends of the sorted array, and set $-\hat{b}_\alpha$ and $-\hat{a}_\alpha$ equal to the resulting endpoints. Then, using $\hat{a}_\alpha$ and $\hat{b}_\alpha$ in equation (2.41), Simar and Wilson find that $(1 - 2\alpha) \cdot 100$ percent of the time,

$$\hat{\theta}_k + \hat{a}_\alpha \leq \theta_k \leq \hat{\theta}_k + \hat{b}_\alpha. \tag{2.43}$$

As a side note, $\hat{a}_\alpha \leq 0$ and $\hat{b}_\alpha \leq 0$ from Theorem 2.11. So the DEA estimate $\hat{\theta}_k$ will lie above the confidence interval calculated from equation (2.43). This is supported by Theorem 2.9, which shows that $\hat{\theta}_k$ is an overestimate of θ_k.

The Proof for Theorem 2.11

To prove Theorem 2.11, we first prove the following Lemma:

Lemma 2.12 *Consider the following two mathematical programs:*

$$\begin{array}{lll} \max\limits_{u,v} & u^T Y_{\cdot,i} & \\ \text{subject to} & v^T X_{\cdot,i} = 1 & \\ & u^T Y \leq v^T X & \\ & u, v \geq 0 & \end{array} \tag{2.44}$$

and

$$\begin{array}{lll} \max\limits_{u,v} & \frac{u^T Y_{\cdot,i}}{v^T X_{\cdot,i}} & \\ \text{subject to} & u^T Y \leq v^T X & \\ & v^T X_{\cdot,i} > 0 & \\ & u, v \geq 0 & \end{array} \tag{2.45}$$

If $\hat{\theta}_i$ is the optimal objective value for model (2.44) (model (2.2)), then $\hat{\theta}_i$ is also the optimal objective value for model (2.45).

Proof: (By contradiction) Let $(\hat{u}, \hat{v})$ be a solution to model (2.44), so that $\hat{\theta}_i = \hat{u}^T Y_{\cdot,i}$. Notice that $(\hat{u}, \hat{v})$ is also feasible for model (2.45).

Now, suppose $(\hat{u}, \hat{v})$ is not a solution to model (2.45). Then there exists some feasible $(\bar{u}, \bar{v})$ with $\bar{\theta}_i = \frac{\bar{u}^T Y_{\cdot,i}}{\bar{v}^T X_{\cdot,i}}$ such that

$$\hat{\theta}_i = \hat{u}^T Y_{\cdot,i} = \frac{\hat{u}^T Y_{\cdot,i}}{\hat{v}^T X_{\cdot,i}} < \frac{\bar{u}^T Y_{\cdot,i}}{\bar{v}^T X_{\cdot,i}} = \bar{\theta}_i. \tag{2.46}$$

Consider the pair

$$(\tilde{u}, \tilde{v}) = \left(\frac{1}{\bar{v}^T X_{\cdot,i}} \bar{u}, \frac{1}{\bar{v}^T X_{\cdot,i}} \bar{v} \right).$$

We have that

$$\tilde{u} \geq 0, \tilde{v} \geq 0$$

since $\bar{u} \geq 0, \bar{v} \geq 0$;

$$\tilde{v}^T X_{\cdot,i} = \frac{1}{\bar{v}^T X_{\cdot,i}} \bar{v}^T X_{\cdot,i} = 1;$$

and

$$\tilde{u}^T Y - \tilde{v}^T X = \frac{1}{\bar{v}^T X_{\cdot,i}} (\bar{u}^T Y - \bar{v}^T X) \leq 0 \Rightarrow \tilde{u}^T Y \leq \tilde{v}^T X$$

since $\bar{u}^T Y \leq \bar{v}^T X$. So, $(\tilde{u}, \tilde{v})$ is feasible for model (2.44). The corresponding objective value is

$$\tilde{u}^T Y_{\cdot,i} = \frac{\bar{u}^T Y_{\cdot,i}}{\bar{v}^T X_{\cdot,i}} = \bar{\theta}_i > \hat{\theta}_i,$$

a contradiction because $\hat{\theta}_i$ is optimal. □

Now we prove the main result:

$$\hat{\theta}_k^* \geq \hat{\theta}_k.$$

Proof: Let $(\hat{u}, \hat{v})$ be a solution to model (2.2) for DMU k, so that $\hat{\theta}_k = \hat{u}^T Y_{\cdot,k}$. Consider the corresponding sampled DEA problem (the dual of model (2.40)),

whose solution yields the objective value $\hat{\theta}_k^*$:

$$\begin{aligned} \max_{u,v} \quad & u^T Y_{\cdot,k} \\ \text{subject to} \quad & u^T Y \leq v^T X^* \\ & v^T X_{\cdot,k} = 1 \\ & u, v \geq 0 \end{aligned} \tag{2.47}$$

where $X_i^* = \frac{\hat{\theta}_i}{\theta_i^*} X_{\cdot,i}$.

We want to show that $(\hat{u}, \hat{v})$ is feasible for model (2.47) to obtain the bound. Notice that the second and third constraints of model (2.47) are satisfied automatically. For the first constraint, consider the following two cases:

1. $\hat{v}^T X_i^* = 0$

   Then

   $$\hat{v}^T X_i^* = \frac{\hat{\theta}_i}{\theta_i^*} \hat{v}^T X_{\cdot,i} = 0 \Rightarrow \hat{v}^T X_{\cdot,i} = 0 \Rightarrow \hat{u}^T Y_{\cdot,i} = 0$$

   from the first constraint of model (2.2) (since $\hat{u} \geq 0$ and $Y_{\cdot,i} \geq 0$). Thus, the first constraint of model (2.47) is satisfied for DMU i.

2. $\hat{v}^T X_i^* > 0$

   Then

   $$\hat{v}^T X_i^* = \frac{\hat{\theta}_i}{\theta_i^*} \hat{v}^T X_{\cdot,i} > 0 \Rightarrow \hat{v}^T X_{\cdot,i} > 0.$$

Thus, the second constraint of model (2.45) is satisfied by $(\hat{u}, \hat{v})$. Because the first and third constraints are also satisfied (by satisfying the constraints of model (2.2)), $(\hat{u}, \hat{v})$ is feasible for model (2.45).

Now by Lemma 2.12, since $\hat{\theta}_i$ is the optimal objective value for model (2.44), it is also the optimal objective value for model (2.45). So,

$$\hat{\theta}_i \geq \frac{\hat{u}^T Y_{\cdot,i}}{\hat{v}^T X_{\cdot,i}}.$$

Thus,

$$\frac{\hat{u}^T Y_{\cdot,i}}{\hat{v}^T X_i^*} = \frac{\theta_i^*}{\hat{\theta}_i} \frac{\hat{u}^T Y_{\cdot,i}}{\hat{v}^T X_{\cdot,i}} \leq \frac{\theta_i^*}{\hat{\theta}_i} \hat{\theta}_i = \theta_i^* \leq 1 \Rightarrow \hat{u}^T Y \leq \hat{v}^T X^*,$$

and so the first constraint of model (2.47) is satisfied for DMU i.

Therefore, for all DMUs, the first constraint of model (2.47) is satisfied, implying that $(\hat{u}, \hat{v})$ is feasible. Because of this, $\hat{\theta}_k^* \geq \hat{\theta}_k$. □

2.3.3 DEA Sensitivity Analysis under Modeling Languages

Modeling languages offer advantages when applying Algorithm 2.2. Under a modeling language, all of the models can be built from one program specification by using parameters in the program to allow for the changing data. Further, there is no need to go outside in order to obtain the new data for each model: we can actually build these data inside the model specification. To actually draw the samples (from Algorithm 2.1), we can use the

predefined probability density functions in GAMS. Finally, because the new input-output data is defined in terms of the old data and the samples, we can use simple parameter re-definitions. In this way, not only can the resulting models be solved, but they can also be defined in GAMS.

Note also that we are not restricted to Simar and Wilson's sensitivity analysis procedure. Any procedure that creates new data points to test against can be used in place of Algorithm 2.1. For example, we could consider Farrell's approach [34] and use a series of production assumptions to obtain additional data points. We could also use a function to model the process that the DMUs go through in order to convert their inputs into their outputs, and obtain (X^*, Y^*) from that function.

In their examples from [97, 98], Simar and Wilson draw 1000 samples; however, they note in [99] that 2000 or more samples may be needed for accurate confidence intervals. As can be seen from Algorithm 2.2, each sample involves a DEA loop consisting of a sequence of linear programs, one for each DMU. Thus, in order to obtain confidence intervals on the efficiency scores, the complete DEA model must be solved 1000 or more times, each time with different data. The slice interface can be used for each of these 1000 samples, making the overall model solution much more efficient. This significantly reduces the solution time and allows us to analyze the scores for DEA models involving large numbers of DMUs.

2.3.4 An Example of DEA Sensitivity

We applied the techniques from [97] to the hospital problem from Section 2.2.4, considering only hospitals at least 10% efficient (this allowed us to eliminate the lower boundary reflection in the density estimate (2.37)). Again, both CPLEX and the slice interface under GAMS were used to solve the resulting DEA problems on different sample sizes. The effect of the sample size on computation time is displayed in Figure 3; these times include the time to generate the samples (done exactly the same way for both solvers) and the time to solve all of the DEA problems resulting from these samples. After 1000 samples, we found it was impractical to continue to use the general CPLEX solver, and so times are only displayed for the CPLEX solver for 500 and 1000 samples. As can be seen from the graph, the slice interface significantly improves overall solution time, enabling us to take more than double the number of samples that we could under CPLEX.

Examining the confidence intervals derived from each set of samples, we found that 500 and even 1000 samples seemed to be too few, with both sets having a number of intervals that were too large or too small in comparison to the larger samples. 1500 and 2000 samples resulted in confidence intervals closer to those found for 2500 samples. These results suggest that we could obtain accurate confidence intervals with 1500, 2000 or 2500 samples but not with 500 or 1000 samples. This further supports the use of slice modeling since the general solver took too long to obtain the results for the larger

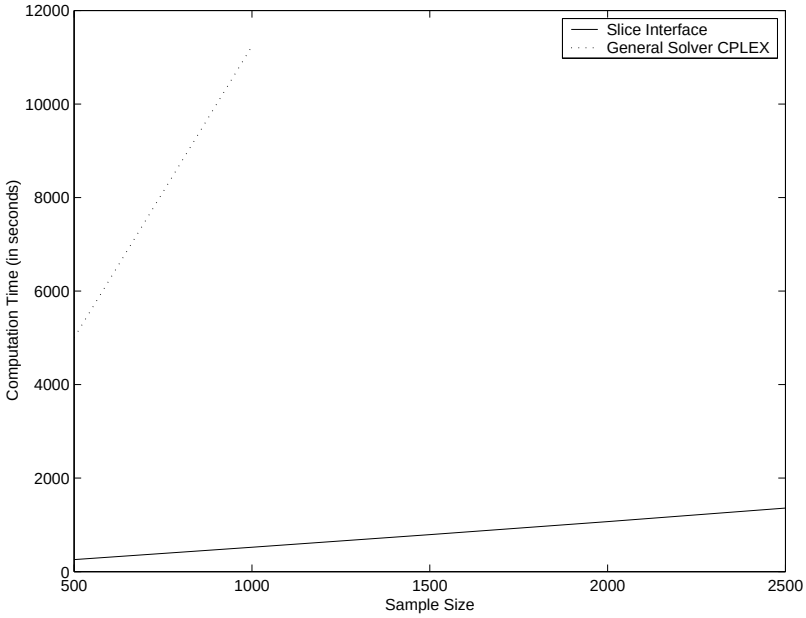


Figure 3: The effect of sample size on computation time in finding confidence intervals for efficiency scores.

samples.

Tables 2 and 3 shows the confidence intervals obtained from drawing 2500 samples. Of the 49 efficient hospitals, only 6 had tight confidence intervals, contained within 90%–100%. 29 others had very wide confidence intervals, with 17 of these spanning over half the efficiency range. The remaining efficient hospitals had confidence intervals around 75%–100%. These results suggest that very few (6 out of 49) of the DEA-efficient hospitals are operating at a truly efficient rate; the remaining DEA-efficient hospitals are very data-dependent.

We conjecture that the use of slice modeling within these applications will

Hospital	$\hat{\theta}_k$	95% CI for θ_k		Hospital	$\hat{\theta}_k$	95% CI for θ_k	
1	1.0000	0.8009	0.9959	27	0.7987	0.7533	0.7958
2	1.0000	0.3747	0.9958	28	0.9049	0.8628	0.9012
3	0.0759	–	–	29	1.0000	0.4047	0.9957
4	1.0000	0.5738	0.9959	30	0.8633	0.7609	0.8598
5	1.0000	0.3941	0.9958	31	0.8462	0.8111	0.8439
6	1.0000	0.7146	0.9960	32	1.0000	0.8951	0.9959
7	1.0000	0.3997	0.9961	33	0.9343	0.8542	0.9300
8	0.9766	0.9103	0.9723	34	1.0000	0.7796	0.9956
9	1.0000	0.6331	0.9958	35	0.7360	0.7040	0.7329
10	0.8486	0.8088	0.8453	36	0.9118	0.8668	0.9081
11	0.9301	0.8572	0.9263	37	1.0000	0.4134	0.9959
12	1.0000	0.7710	0.9962	38	1.0000	0.3470	0.9955
13	1.0000	0.3651	0.9959	39	1.0000	0.4653	0.9962
14	0.9847	0.9379	0.9805	40	0.8321	0.7496	0.8287
15	0.9862	0.8607	0.9821	41	0.9381	0.8569	0.9342
16	1.0000	0.3733	0.9956	42	0.9379	0.8132	0.9339
17	1.0000	0.7755	0.9957	43	1.0000	0.8778	0.9957
18	0.9106	0.8640	0.9082	44	0.2054	0.1991	0.2049
19	1.0000	0.4072	0.9958	45	1.0000	0.3651	0.9958
20	0.6225	0.6023	0.6209	46	1.0000	0.6005	0.9956
21	1.0000	0.8738	0.9958	47	1.0000	0.4201	0.9964
22	0.9045	0.8310	0.9004	48	0.0473	–	–
23	1.0000	0.7806	0.9958	49	1.0000	0.8951	0.9960
24	1.0000	0.9518	0.9957	50	0.6349	0.5329	0.6324
25	0.9834	0.8756	0.9794	51	0.9664	0.9023	0.9621
26	0.9895	0.9141	0.9854	52	1.0000	0.7844	0.9960

Table 2: Confidence intervals for hospitals 1–52.

Hospital	$\hat{\theta}_k$	95% CI for θ_k		Hospital	$\hat{\theta}_k$	95% CI for θ_k	
53	1.0000	0.9466	0.9958	79	0.5121	0.4413	0.5103
54	0.8937	0.8425	0.8909	80	1.0000	0.8818	0.9960
55	0.8780	0.7342	0.8744	81	0.8067	0.7121	0.8042
56	0.9418	0.8659	0.9377	82	0.9497	0.8316	0.9458
57	0.6271	0.5932	0.6252	83	1.0000	0.3381	0.9958
58	0.8471	0.7972	0.8436	84	0.8727	0.8019	0.8690
59	0.9943	0.9004	0.9902	85	0.8717	0.8146	0.8679
60	0.8953	0.8012	0.8914	86	0.9104	0.8264	0.9069
61	0.9196	0.7769	0.9156	87	1.0000	0.9499	0.9961
62	1.0000	0.8113	0.9960	88	0.8791	0.8357	0.8753
63	1.0000	0.6652	0.9958	89	1.0000	0.5169	0.9957
64	0.9449	0.8947	0.9410	90	0.8148	0.7074	0.8115
65	0.9690	0.9389	0.9664	91	0.8410	0.7944	0.8376
66	0.7712	0.7384	0.7681	92	1.0000	0.6604	0.9959
67	0.9753	0.9326	0.9717	93	1.0000	0.9079	0.9955
68	0.6963	0.6467	0.6944	94	1.0000	0.8554	0.9957
69	1.0000	0.6564	0.9954	95	0.9522	0.8974	0.9485
70	0.9042	0.8658	0.9002	96	0.7874	0.7162	0.7842
71	1.0000	0.4671	0.9959	97	1.0000	0.3702	0.9958
72	0.9485	0.9014	0.9442	98	0.7959	0.7601	0.7937
73	0.6970	0.6442	0.6949	99	1.0000	0.4089	0.9956
74	0.9620	0.9004	0.9582	100	1.0000	0.8133	0.9958
75	1.0000	0.7981	0.9956	101	1.0000	0.3713	0.9958
76	0.9452	0.7472	0.9422	102	1.0000	0.6140	0.9957
77	1.0000	0.6410	0.9960	103	1.0000	0.8979	0.9955
78	1.0000	0.5638	0.9955	104	1.0000	0.3935	0.9957

Table 3: Confidence intervals for hospitals 53–104.

make such analyses commonplace. Since the slice interface has been available for download from the GAMS web site at [41], we have been contacted by a few interested individuals. One individual, in particular, was successful in determining DEA confidence intervals for his application using the slice interface to implement Algorithm 2.2. He informed us that the slice interface would be used in the future in place of other DEA programs [43].

2.4 Slice Models for Cross-Validation

DEA models are examples of addition slice models: to define a particular problem, slices are added to the core model. Other types of slice models exist, where slices are *deleted* from the core model. Cross-validation models are examples of these deletion slice models.

The technique of cross-validation measures a prediction model's error using only the available data set to train and test the model. In k-fold cross-validation [32], the available data set is divided into k (generally about equal) pieces. Then the model is trained k times, each time leaving one of the k pieces out of the training set and using it as the testing set instead. Performing cross-validation on prediction models involving mathematical programming can be regarded as another example of slice modeling because the resulting models are programs with the same structure but different data.

But, unlike DEA models in which slices are added to the core data, cross-validation models require that slices be deleted from the core (training) data: the k-th individual program can be defined by deleting the k-th piece. In this case, the "slicing" is done by elimination: everything *except* the particular slice is included in the program.

In this section, we discuss cross-validation and apply the technique to examples for breast cancer diagnosis. Then, we show how these models can be implemented in GAMS under the slice interface, providing results for the examples discussed.

2.4.1 Cross-Validation in Applications

Cross-validation is used widely in many studies to test a model's validity. Recently, cross-validation was used in a feature selection model to identify regulatory elements in gene expression data [57]. Ejrnaes et al. [33] used cross-validation to guide their development of a neural network for predicting the probability of potential conservation interest in habitats. Hargreaves and Annan [45] used cross-validation to show that their climate model, using a Monte Carlo Markov Chain method, will forecast ocean temperature, atmospheric carbon dioxide concentration and global ice volume reasonably well.

In addition to testing validity, cross-validation is also used to compare models. Dolan et al. [29] used cross-validation to compare spatial prediction

models developed using the method of quasi-likelihood. Tempelman and Gianola [100] developed two models for predicting fertility in dairy cattle, and used cross-validation to support the use of their models over another (linear) model. Similarly, Lee and Mangasarian [62] used cross-validation correctness on publicly available databases as a measure of how well their smooth support vector machine (SSVM) performs in comparison to four other methods.

2.4.2 Cross-Validation and Support Vector Machines

We focus on an example involving binary classification. We make use of a support vector machine, trained on a set of data for which the outcome (either $+1$ or -1) is already known. In order to test the classifier, the model makes use of cross-validation, leaving a portion of the data out of the training for use as testing data.

We consider the standard linear support vector machine model from [9]:

$$\begin{aligned} \min_{w,\gamma,z} \quad & \tfrac{1}{2}\|w\|_2^2 + Ce^T z \\ \text{subject to} \quad & D(Aw - \gamma) + z \geq e, \;\; z \geq 0 \end{aligned} \tag{2.48}$$

Here, A is a matrix containing the training data (subjects by features) and D is a diagonal matrix with values ± 1, denoting the two classification labels. C is a parameter weighting the importance of maximizing the margin between the classes versus minimizing the misclassification error (z). e is a vector of

ones. The solution (w, γ) is used to define a separating hyperplane $\{x|w^T x = \gamma\}$ that divides the two classes from each other.

Prior to solving the quadratic programming (QP) model (2.48), we first consider a modified version that can be converted into a linear program:

$$\begin{array}{ll} \min\limits_{w,\gamma,z} & \|w\|_1 + Ce^T z \\ \text{subject to} & D(Aw - \gamma) + z \geq e, \;\; z \geq 0 \end{array} \tag{2.49}$$

Model (2.49) replaces the Euclidean-norm margin measurement of (2.48) with the sup-norm measurement (resulting in the minimization of the 1-norm) and can be converted into a linear program by adding additional variables, y:

$$\begin{array}{ll} \min\limits_{w,\gamma,z,y} & e^T y + Ce^T z \\ \text{subject to} & D(Aw - \gamma) + z \geq e, \;\; z \geq 0 \\ & y \geq w \geq -y \end{array} \tag{2.50}$$

2.4.3 Cross-Validation Models as Slice Models

Models (2.48) and (2.50) are not slice models per se. They become slice models under cross-validation, where they are solved multiple times on different pieces of data. In these cases, only the data A and D vary between solves, appropriately fitting the definition of slice models. In terms of model (1.1), this implies that the slices *removed* from the full model are:

$$D_{k,k}(A_{k,\cdot}w + z_k) \geq 1 \tag{2.51}$$

where k here represents the set of data for the k-th cross-validation test.

We use the data from the inequalities (2.51) to evaluate the model built during the cross-validation training phase. Let $\overline{w}$ and $\overline{\gamma}$ represent the optimal values for the k-th model. To determine the how accurately the k-th model predicts, we count the number of missed test cases. This is done by counting then number of times that a test point ends up on the wrong side of the separating hyperplane. Mathematically, this is expressed as

$$D_{k,k}(A_{k,\cdot}\overline{w} - \overline{\gamma}) \leq 0. \tag{2.52}$$

2.4.4 Support Vector Machine Applications

Deletion style problems are defined similarly to the addition style problems under the slice interface. Figure 4 shows the original GAMS formulation for 10-fold cross-validation applied to (2.50). Only training is carried out in this model, displaying values that define the separating hyperplane (w, γ) for later testing. The equations are defined over the training sets, generated dynamically from the testing sets inside the solve loop. The testing sets are defined by calling the batch include file `gentestset.inc` (Figure 5). A and D are defined in `wdbc.gms`.

Figure 6 shows the slice formulation for the same model (where identical lines above the problem definition have been omitted). The major differences between the original and slice formulations are the lack of the solve loop and training sets in the latter. Also, the slice solver is used with an options file

```
$title Ten-fold cross-validation example
$eolcom !

$setglobal num_folds 10
set p /1*%num_folds%/;    ! folds to perform
! set i /1*%num_entries/; ! entries (from data file)
! set k /1*%num_feat%/;   ! features (from data file)

! Read in data
$include "wdbc.gms"
parameter C /1/;

! Declare testing and training sets
set test(i);
set trai(i);

! Define problem
positive variables z(i), y(k);
variables obj, w(k), gamma;

equations obj_def, sep_def(i), bd1(k), bd2(k);
obj_def..  obj =e= C*sum(trai, z(trai)) + sum(k,y(k));
sep_def(trai)..  D(trai)*(sum(k, A(trai,k)*w(k))
                          - gamma) + z(trai) =g= 1;
bd1(k)..   y(k) =g= -w(k);
bd2(k)..   y(k) =g= w(k);

model train /all/;

! Solve
loop(p,
! Generate testing and training sets
$batinclude gentestset.inc i
trai(i) = not test(i);
solve train using lp minimizing obj;
display w.l,gamma.l;
);
```

Figure 4: The original GAMS formulation for cross-validation applied to model (2.50).

```
$set i_num floor(card(i) / %num_folds%)

test(%1) = no;

test(%1)$((ord(i) > %i_num%*(ord(p)-1)) and
            (ord(i) <= %i_num%*ord(p))) = yes;
! put leftovers in last set
test(%1)$((ord(p) eq %num_folds%) and
            (ord(i) > %i_num%*%num_folds%)) = yes;
```

Figure 5: The batch include file `gentestset.inc` for generating the testing sets.

`dea.opt` (Figure 7), which tells the interface to delete slices rather than add them (`slicetype 0`).

The solve loop is eliminated in the slice formulation by the way in which the interface handles the data. Here again, all of the data is passed to the interface initially and the key name `slice` is used to determine which equations/data belong to which problems. The training sets are eliminated from the formulation since they are defined implicitly by deleting in turn each testing set from the complete collection of testing sets. In the slice formulation, the testing sets are defined prior to the solve statement (since all of the data is passed to the interface at once), and are indexed by `p` (the fold set). In addition, the equations `sep_def` that depend upon the testing sets are indexed by `p`. By creating a mapping from the slice/fold set `p` to a subset of the patient set `i`, the corresponding (testing) equations are deleted from each of the problems in turn.

```
! Define problem
alias(p,slice);
positive variables z(i), y(k);
variables obj, w(k), gamma;

equations obj_def, sep_def(i,slice), bd1(k), bd2(k);
obj_def..  obj =e= C*sum(i, z(i)) + sum(k,y(k));
sep_def(i,p)$test(i,p)..  D(i)*(sum(k, A(i,k)*w(k))
                                - gamma) + z(i) =g= 1;
bd1(k)..   y(k) =g= -w(k);
bd2(k)..   y(k) =g= w(k);

model train /all/;

! Solve
option lp = dea;  train.optfile = 1;

! Generate testing sets (to be deleted in each problem)
loop(p,
$batinclude gentestset.inc "i,p"
);

solve train using lp minimizing obj;
```

Figure 6: The slice model formulation for cross-validation applied to model (2.50).

```
slicetype 0
eqnchk 0
primval w,gamma
```

Figure 7: The options file for the slice model formulation.

Note that equation `obj_def` changes from a sum over the training set to a sum over the entire set in the slice formulation. Variables related to the testing sets in `obj_def` will be eliminated (since they do not appear elsewhere in the model). By default, the slice interface ensures that the number of equations in each problem is the same; most slice models have this property. However, in cross-validation, `card(p)` may not evenly divide `card(i)`. The last problem may have fewer equations than the rest, so we use the option `eqnchk` to turn off the automatic equation check.

For the QP problem (2.48), we extend GAMS/QPWRAP [42]. Under GAMS/QPWRAP, a QP model is formulated as a combination of a linear program (specified in the GAMS model) and a Q matrix (specified in a text file). GAMS/QPWRAP passes the Q matrix to the solver.

Figure 8 shows the GAMS formulation for 10-fold cross-validation applied to model (2.48) (identical lines above the problem definition have been omitted). This formulation is very similar to that of Figure 6, with `slice` identifying the slice constraints. It uses the same options file (Figure 7). Only the linear portion of model (2.48) is given explicitly; the quadratic portion is specified by the Q matrix and written to the file `qmatrix.txt` prior to the solver call. In addition, the solver has been changed from `dea` to `deaqp`, that calls GAMS/QPWRAP to read the Q matrix prior to calling the slice interface.

Since the file `qmatrix.txt` must exist prior to the solver call, the ability

```
! Define the Q matrix
parameter q(k,k);  q(k,k) = .5;

! Now write the Q matrix to a file named qmatrix.txt
file qp / qmatrix.txt /;
qp.pc=5; qp.nr=2; qp.nd=13; qp.nw=0; ! formatting options

put qp 'Q Matrix for svm';
loop(k, put / 'w' k.tl 'w' k.tl  q(k,k))
putclose;

! Define problem
alias(p,slice);
positive variables z(i);
variables obj, w(k), gamma;

equations obj_def, sep_def(i,slice);
obj_def..  obj =e= C*sum(i, z(i));
sep_def(i,p)$test(i,p)..  D(i)*(sum(k, A(i,k)*w(k))
                                - gamma) + z(i) =g= 1;

model train /all/;

! Solve
option lp = deaqp;  train.optfile = 1;

! Generate testing sets (to be deleted in each problem)
loop(p,
$batinclude gentestset.inc "i,p"
);

solve train using lp minimizing obj;
```

Figure 8: The slice model formulation for cross-validation applied to model (2.48).

for the slice interface to solve QP slice models is limited. The slice interface can only solve QP models where the Q matrix does not change between solves, that is, where the Q matrix is part of the core data.

For a cross-validation application, we focus on an example involving the classification of patients' breast tumors as either malignant or benign. For the data, we use the Wisconsin Diagnosis Breast Cancer Database (WDBC) [109], consisting of 569 patients and 30 features.

On the WDBC data set, we ran all-but-one testing on both the linear and the quadratic formulations. For the linear model (2.50), testing time was cut by more than half: under GAMS/CPLEX, testing took 541.71 seconds; under the slice interface, testing took 269.03 seconds. Testing accuracy was 95.08%.

Since the quadratic model uses a barrier code, previous solutions are not very useful in reducing computational times. For completeness, we cite the following results. Testing of model (2.48) took 7403.03 seconds under GAMS/CPLEXQP and 7221.74 seconds under QP slice interface; testing accuracy was 95.42%. However, much faster results can be obtained under nonlinear solvers. Under GAMS/CONOPT, testing of model (2.48) took 519.3 seconds; under GAMS/PATH, testing took 691.9 seconds. Eliminating regeneration time from these results, a slice approach would improve these times to around 343.18 seconds and 245.45 seconds, respectively.

Chapter 3

Model Building: Likelihood Basis Pursuit

A large body of research focuses on medical applications involving large databases of raw data. A common goal in such applications is to use the raw data for classification purposes in order to predict outcomes; another goal is variable selection (feature selection in machine language terminology), where important variables related to the outcome are identified. In classification, we use observations (such as blood pressure, height, weight, age, etc.) to classify patients into classes related to the outcomes (such as recurrent cancer or not, disease susceptible or not, death likely or not, etc.). In variable selection, we look for those observations which influence the outcome the most. A variety of models have been developed to achieve these goals.

In Section 2.4.2, we discussed the standard linear support vector machine (2.48) and a linear programming version (2.50). These models look for

a linear classifier

$$\{x|w^T x = \gamma\}$$

that distinguishes datapoints in one class (+1) from datapoints in another class (−1). Nonlinear separators can also be found through the use of kernels. Taking the Wolfe dual [74, Section 8.2] of (2.48), we obtain

$$\begin{array}{ll} \min\limits_{u} & \frac{1}{2}u^T DAA^T Du - e^T u \\ \text{subject to} & e^T Du = 0 \\ & 0 \leq u \leq Ce \end{array} \tag{3.1}$$

Then replacing AA^T in the objective function of (3.1) with a nonlinear kernel $K(A, A^T)$ [77] where $K : \mathbb{R}^{m\times n} \times \mathbb{R}^{n\times m} \rightarrow \mathbb{R}^{m\times m}$, we obtain

$$\begin{array}{ll} \min\limits_{u} & \frac{1}{2}u^T DK(A, A^T)Du - e^T u \\ \text{subject to} & e^T Du = 0 \\ & 0 \leq u \leq Ce \end{array} \tag{3.2}$$

From the solution of (3.2), we obtain the nonlinear separator

$$\{x|K(x^T, A^T)Du = \gamma\}.$$

Variations on these models are discussed in [9, 37, 77], among others. We note that in the above formulation, Mercer's condition [102] of positive definiteness on $K(A, A^T)$ is not needed.

Other approaches make use of statistical methods to predict outcomes. For example, consider a linear model which looks for coefficients b such that, for the i-th observation,

$$y_i = b^T x_i + \varepsilon_i. \tag{3.3}$$

Here, y_i is the outcome, x_i is the vector of observations, and ε_i represents random variations. Traditionally, b is found by a least squares fit, but link functions that describe how the expected outcome is related to $b^T x_i$ can also be used to define generalized linear models [53].

The approaches discussed thus far focus on classification. However, they can be modified to also consider variable selection. For example, for the 1-norm linear support vector machine (2.50), we can find the k most influential observations by restricting w to be nonzero in only k components; this results in a mixed-integer program. (In fact, model (2.50) itself has excellent feature selection properties [15, Table 1].) We could put similar restrictions on b in the linear model (3.3). Another approach for the linear model (3.3) measures the importance of each observation j after (full) classification by considering the averaged norm of $b_j x_{\cdot,j}$ (see [118], where this is done for observations under a more complicated model).

In this chapter, we consider a statistical approach called likelihood basis pursuit (LBP). Unlike support vector machines that look for a classifier, LBP derives a probability estimate for the outcome by maximizing the penalized likelihood of a nonlinear model. The penalty terms come from basis pursuit

ideas and use the 1-norm of the model's variables. In the next section, we discuss the LBP procedure and two resulting models, a main effects model and a two-factor interaction model. We then discuss fitting the model by selecting appropriate parameters under a grid search. We show how this procedure can be improved through the use of slice modeling techniques. Finally, we describe alternate procedures for parameter selection that make use of derivative-free optimization algorithms. Given appropriate starting information, we show that these algorithms can obtain better solutions than the grid search with fewer function evaluations.

3.1 Background for Likelihood Basis Pursuit

The ultimate goal of the LBP models is to estimate probability. In practice though, we do not look for a direct probability measure but rather approach it by measuring the log odds ratios.

3.1.1 The Function to Estimate: Log Odds Ratio

We begin with a training set of n observations, (x_i, y_i) $(i = 1, \ldots, n)$, where

$$x_i = (x_i^1, \ldots, x_i^D) \in \mathbb{R}^D$$

is the explanatory vector and

$$y_i \in \{0, 1\}$$

is the classification label (typically 1 represents occurrence of some event, and 0 represents non-occurrence). For any explanatory vector, x, we would like to determine the corresponding label y (either 1 or 0). If we knew

$$p(x) = \Pr\{y = 1|x\},$$

then the label could be determined by applying Bayes' rule for minimizing the expected misclassification rate [67]:

$$y = \begin{cases} 1 & \text{if } p(x) > \frac{1}{2} \\ \\ 0 & \text{otherwise.} \end{cases}$$

Our goal is to estimate $p(x)$, using the training data (x_i, y_i). We do this with logistic regression. Logistic regression is useful in this case because it transforms linear regression results, which are values between positive and negative infinity, to values between 0 and 1 [86] — ideal values for a probability measure.

However, rather than working directly with the probability p, we instead focus on the log odds ratio. The odds ratio is a value associated with each explanatory variable that compares the probability of occurrence to the probability of non-occurrence. It describes the relative amount by which the outcome improves or declines when the value of that explanatory variable is increased by one unit [86]. We specifically consider the logarithm of the odds ratio:

$$f(x) = \log\left(\frac{p(x)}{1 - p(x)}\right).$$

Notice that p can be recovered from f by:

$$p(x) = \frac{\exp(f(x))}{1 + \exp(f(x))}.$$

We use smoothing spline analysis of variance (SS-ANOVA) models to model f. SS-ANOVA is a general technique for building multivariate, non-parametric regression models. The idea behind these models is to decompose a function into components that satisfy generalized ANOVA side conditions [106]. Specifically, we search for f in a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ by considering the decomposition [117, 118]:

$$f = d_0 + \sum_{\alpha=1}^{D} f_\alpha(x^\alpha) + \sum_{\beta>\alpha} f_{\alpha\beta}(x^\alpha, x^\beta) + \text{ higher-order interactions.} \quad (3.4)$$

Here, d_0 is constant, f_α's are main effects, and $f_{\alpha\beta}$'s are two-factor interactions. Following [117, 118], we assume that $f_\alpha \in \mathcal{H}^{(\alpha)}$, where $\mathcal{H}^{(\alpha)}$ is a RKHS generated by some specified kernel. For the interaction terms, $f_{\alpha\beta} \in \mathcal{H}^{(\alpha)} \otimes \mathcal{H}^{(\beta)}$, where $\otimes$ denotes the tensor product product. Then

$$\mathcal{H} = [1] \oplus \sum_{\alpha=1}^{D} \mathcal{H}^{(\alpha)} \oplus \sum_{\beta>\alpha} [\mathcal{H}^{(\alpha)} \otimes \mathcal{H}^{(\beta)}] \oplus \cdots .$$

Following [117, 118], $\mathcal{H}^{(\alpha)}$ can be decomposed into a (finite-dimensional) parametric part and a smooth part (the orthogonal complement of the parametric part):

$$\mathcal{H}^{(\alpha)} = \mathcal{H}_\pi^{(\alpha)} \oplus \mathcal{H}_s^{(\alpha)}.$$

Similarly,

$$\mathcal{H}^{(\alpha)} \otimes \mathcal{H}^{(\beta)} = [\mathcal{H}_\pi^{(\alpha)} \otimes \mathcal{H}_\pi^{(\beta)}] \oplus [\mathcal{H}_\pi^{(\alpha)} \otimes \mathcal{H}_s^{(\beta)}] \oplus [\mathcal{H}_s^{(\alpha)} \otimes \mathcal{H}_\pi^{(\beta)}] \oplus [\mathcal{H}_s^{(\alpha)} \otimes \mathcal{H}_s^{(\beta)}].$$

Thus, we obtain an orthogonal decomposition for $\mathcal{H}$ into sums of products of finite-dimensional parametric spaces, plus smooth main effects subspaces, plus two-factor interactions spaces (of forms parametric $\otimes$ parametric, smooth $\otimes$ parametric, and smooth $\otimes$ smooth), and higher-order interaction subspaces [117, 118]. This decomposition aids us in defining the form for f. To make the model more manageable, we can truncate the higher-order interaction terms. Then, $\mathcal{H}$ is the direct sum of a finite number of component subspaces, say Q. If R_l denotes the kernel of each component subspace, then

$$\mathcal{H} = \oplus_{l=1}^{Q} \text{span}\{R_l\}.$$

When n is large, searching $\mathcal{H}$ can be computationally intensive. To reduce the load, we limit the number of basis functions for the components of $\mathcal{H}$. In [118], it is noted that the number of basis terms can be much smaller than n without degrading performance. As in [118], we apply a random sampling to draw $N \leq n$ observations, denoted $\{x_{1*}, \ldots, x_{N*}\}$, for use in defining $\mathcal{H}$. (Other methods for choosing the N observations include clustering [112].) Then, rather than searching $\mathcal{H}$ for f, we search an approximated function space

$$\mathcal{H}_* = \oplus_{l=1}^{Q} \text{span}\{R_l(x_{j*}, \cdot), j = 1, \ldots, N\}.$$

Initially, we consider only the main effects. This results in dropping two-factor and higher-order interaction terms from consideration in equation (3.4) and truncating $\mathcal{H}_*$ to direct sums of products of parametric spaces

plus smooth effects subspaces:

$$\mathcal{H} = \oplus_{\alpha=1}^{D} \text{span}\{k_1(x^\alpha), K_1(x^\alpha, x_{j*}^\alpha), j = 1, \ldots, N\}.$$

Here k_1 represents the (one-dimensional) parametric portion and K_1 represents the smooth portion. For our examples, we take $m = 2$ in [104, equation (10.2.4)] to obtain:

$$k_1(t) = t - \frac{1}{2} \tag{3.5}$$

and

$$K_1(s,t) = k_2(s)k_2(t) - k_4(|s-t|), \tag{3.6}$$

where

$$k_2(t) = \frac{1}{2}\left(k_1^2(t) - \frac{1}{12}\right)$$

and

$$k_4(t) = \frac{1}{24}\left(k_1^4(t) - \frac{1}{2}k_1^2(t) + \frac{7}{240}\right).$$

Then, the main effects model, also referred to as the "additive" model, for f becomes:

$$f(x) = d_0 + \sum_{\alpha=1}^{D} d_\alpha(x^\alpha - \frac{1}{2}) + \sum_{j=1}^{N}\sum_{\alpha=1}^{D} C_{j,\alpha}K_1(x^\alpha, x_{j*}^\alpha). \tag{3.7}$$

Adding the two-factor interaction effects, the parametric portion now consists of the parametric main effects $\{k_1(x^\alpha)|\alpha = 1, \ldots, D\}$ and the parametric-parametric interaction terms $\{k_1(x^\alpha)k_1(x^\beta)|\alpha = 1, \ldots, D; \beta > \alpha\}$. The

smooth portion consists of the smooth main effects, the parametric-smooth effects and the smooth-smooth effects. Together, this yields the "full" model:

$$\begin{aligned} f(x) &= d_0 + \sum_{\alpha=1}^{D} d_\alpha (x^\alpha - \frac{1}{2}) + \sum_{\alpha=1}^{D} \sum_{\beta>\alpha} d_{\alpha,\beta} (x^\alpha - \frac{1}{2})(x^\beta - \frac{1}{2}) \\ &+ \sum_{j=1}^{N} \sum_{\alpha=1}^{D} C^s_{j,\alpha} K_1(x^\alpha, x^\alpha_{j*}) \\ &+ \sum_{j=1}^{N} \sum_{\alpha=1}^{D} \sum_{\alpha\neq\beta} C^{\pi s}_{j,\alpha,\beta} K_1(x^\alpha, x^\alpha_{j*})(x^\beta_j - \frac{1}{2})(x^\beta_{j*} - \frac{1}{2}) \\ &+ \sum_{j=1}^{N} \sum_{\alpha=1}^{D} \sum_{\beta>\alpha} C^{ss}_{j,\alpha,\beta} K_1(x^\alpha, x^\alpha_{j*}) K_1(x^\beta, x^\beta_{j*}) \end{aligned} \tag{3.8}$$

For notational convenience, we let

$$f_i = f(x_i).$$

We also let

$$[f_1, \ldots, f_n]^T = f = Td + KC. \tag{3.9}$$

where T contains the parametric effects and K contains the smooth effects. T and K will differ depending upon whether we are considering the additive model or the full model.

Once values for the d and C variables are determined in either the additive or the full model, $f(x)$ and $p(x)$ can be found. To find the d and C values, we maximize likelihood over the training data. Likelihood is given by:

$$L = \prod_{i=1}^{n} p(x_i)^{y_i} (1 - p(x_i))^{1-y_i}.$$

Here again, we are actually interested in the corresponding logarithm:

$$\begin{aligned}
\log(L) &= \sum_{i=1}^{n} \log\left[p(x_i)^{y_i}(1-p(x_i))^{1-y_i}\right] \\
&= \sum_{i=1}^{n} \left[y_i \log(p(x_i)) + (1-y_i)\log(1-p(x_i))\right] \\
&= \sum_{i=1}^{n} \left[y_i \left[\log(p(x_i)) - \log(1-p(x_i))\right] + \log(1-p(x_i))\right] \\
&= \sum_{i=1}^{n} \left[y_i \log\left(\frac{p(x_i)}{1-p(x_i)}\right) - \log\left(\frac{1}{1-p(x_i)}\right)\right] \\
&= \sum_{i=1}^{n} \left[y_i f_i - \log\left(1 + \frac{p(x_i)}{1-p(x_i)}\right)\right] \\
&= \sum_{i=1}^{n} \left[y_i f_i - \log\left(1+e^{f_i}\right)\right]. \qquad (3.10)
\end{aligned}$$

Our goal is to maximize likelihood, thus we look to maximize $\log(L)$ with respect to d and C. We would like to find the "best" solution, where we have some definition for what constitutes the "best" solution. We follow basis pursuit ideas and define the "best" to mean the solution where the d and C variables have the smallest 1-norms.

3.1.2 Basis Pursuit

Basis pursuit was originally developed by Chen et al. [25] for signal decomposition in overcomplete dictionaries. In the environment of overcomplete dictionaries, multiple decompositions can exist for the same signal. Basis pursuit offers a way to find a "best" decomposition, where Chen et al. [25]

defined "best" to mean the solution whose coefficients had the smallest 1-norms. Using 1-norms, Chen et al. [25] were able to formulate linear programming problems to find a single ("the best") signal decomposition. Not only are these signal decomposition problems under basis pursuit "easy" to solve in comparison to other methods since they use linear programming, but they also result in sparse decompositions.

Extending basis pursuit ideas to SS-ANOVA decomposition, Zhang et al. [117, 118] penalize the log-likelihood function with the 1-norms of the variables d and C. This allows us to take advantage of the sparsity that Chen et al. [25] observed for signal decompositions — under basis pursuit ideas, we typically find sparse solutions. Since we obtain a sparser solution than we might otherwise, we also have some idea of which features are most important in influencing the outcomes. In [118], the importance of a particular input is measured by the size of its functional 1-norm, calculated as the average of the function values estimated at all the data points. For the main effect, we have

$$L_1(f_\alpha) = \frac{1}{n}\sum_{i=1}^{n} |f_\alpha(x_i^\alpha)| = \frac{1}{n}\sum_{i=1}^{n} \left| d_\alpha k_1(x_i^\alpha) + \sum_{j=1}^{N} C_{\alpha,j} K_1(x_i^\alpha, x_{j*}^\alpha) \right|$$

(for $\alpha = 1, \ldots, d$) and for the two-factor interactions, we have

$$\begin{aligned} L_1(f_{\alpha,\beta}) &= \tfrac{1}{n}\textstyle\sum_{i=1}^{n} |f_{\alpha,\beta}(x_i^\alpha, x_i^\beta)| \\ &= \tfrac{1}{n}\textstyle\sum_{i=1}^{n} \Big| d_{\alpha,\beta} k_1(x_i^\alpha) k_1(x_i^\beta) + \sum_{j=1}^{N} C_{j,\alpha,\beta}^{\pi s} K_1(x_i^\alpha, x_{j*}^\alpha) k_1(x_i^\beta) k_1(x_{j*}^\beta) \\ &\quad + \textstyle\sum_{j=1}^{N} C_{j,\beta,\alpha}^{\pi s} K_1(x_i^\beta, x_{j*}^\beta) k_1(x_i^\alpha) k_1(x_{j*}^\alpha) \\ &\quad + \textstyle\sum_{j=1}^{N} C_{j,\alpha,\beta}^{ss} K_1(x_i^\alpha, x_{j*}^\alpha) K_1(x_i^\beta, x_{j*}^\beta) \Big| \end{aligned}$$

(for $\beta < \alpha$). The larger the functional 1-norm, the more important that input is. (An equivalent measure was found to be the functional 2-norm.) When d, C are zero under the solution, we have no contribution to the corresponding L_1 value, resulting in a smaller L_1 and hence a less important input. Thus, applying basis pursuit to the maximum likelihood problem from equation (3.10) enhances our ability to perform variable selection.

3.1.3 LBP Models

Combining the basis pursuit penalty terms with the log likelihood objective function (3.10), we obtain the complete LBP models:

$$\min_{d,C} \frac{1}{n} \sum_{i=1}^{n} \left[-y_i f_i + \log\left(1 + e^{f_i}\right) \right] + \lambda_\pi \sum_{\alpha=1}^{D} |d_\alpha| + \lambda_s \sum_{j=1}^{N} \sum_{\alpha=1}^{D} |C_{j,\alpha}| \qquad (3.11)$$

for the additive model with f given by (3.7), or

$$\begin{aligned}\min_{d,C} \quad & \frac{1}{n}\sum_{i=1}^{n}\left[-y_i f_i + \log\left(1+e^{f_i}\right)\right] + \lambda_\pi \sum_{\alpha=1}^{D}|d_\alpha| + \lambda_{\pi\pi}\sum_{\alpha=1}^{D}\sum_{\beta>\alpha}|d_{\alpha,\beta}| \\ + \quad & \lambda_s \sum_{j=1}^{N}\sum_{\alpha=1}^{D}|C^s_{j,\alpha}| + \lambda_{\pi s}\sum_{j=1}^{N}\sum_{\alpha=1}^{D}\sum_{\alpha\neq\beta}|C^{\pi s}_{j,\alpha,\beta}| \\ + \quad & \lambda_{ss}\sum_{j=1}^{N}\sum_{\alpha=1}^{D}\sum_{\beta>\alpha}|C^{ss}_{j,\alpha,\beta}| \end{aligned} \tag{3.12}$$

for the full model with f given by (3.8). In these programs, λ_π, $\lambda_{\pi\pi}$, λ_s, $\lambda_{\pi s}$, and λ_{ss} are regularization parameters that balance the trade-off between maximizing likelihood and minimizing the penalty terms.

To solve, we replace the absolute value terms with nonnegative variables constrained to be the absolute values of the d and C variables, as in [25] or as was done for the 1-norm linear support vector machine (2.50). This results in programs with nonlinear objective functions and linear constraints. The additive model (3.11) becomes:

$$\min_{d,C,\xi,\eta} \quad \frac{1}{n}\sum_{i=1}^{n}\left[-y_i f_i + \log\left(1+e^{f_i}\right)\right] + \lambda_\pi e^T\xi + \lambda_s e^T\eta$$

subject to

$$\begin{aligned} & \xi_\alpha \geq d_\alpha, \qquad \xi_\alpha \geq -d_\alpha \qquad \forall\alpha \\ & \eta_{j,\alpha} \geq C_{j,\alpha}, \quad \eta_{j,\alpha} \geq -C_{j,\alpha} \quad \forall j,\alpha \\ & \xi \geq 0, \eta \geq 0 \end{aligned} \tag{3.13}$$

where f is defined in (3.7). The full model (3.12) becomes

$$\begin{aligned}\min_{d,C,\xi,\tau,\eta^s,\eta^{\pi s},\eta^{ss}} \quad & \frac{1}{n}\sum_{i=1}^{n}\left[-y_i f_i + \log\left(1+e^{f_i}\right)\right] + \lambda_\pi e^T\xi + \lambda_{\pi\pi}e^T\tau \\ & +\lambda_s e^T\eta^s + \lambda_{\pi s}e^T\eta^{\pi s} + \lambda_{ss}e^T\eta^{ss}\end{aligned}$$

subject to

$$\begin{array}{llll}
\xi_\alpha \geq d_\alpha, & \xi_\alpha \geq -d_\alpha & \forall\alpha \\
\tau_{\alpha,\beta} \geq d_{\alpha,\beta}, & \tau_{\alpha,\beta} \geq -d_{\alpha,\beta} & \forall\alpha,\beta \\
\eta^s_{j,\alpha} \geq C^s_{j,\alpha}, & \eta^s_{j,\alpha} \geq -C^s_{j,\alpha} & \forall j,\alpha \\
\eta^{\pi s}_{j,\alpha,\beta} \geq C^{\pi s}_{j,\alpha,\beta}, & \eta^{\pi s}_{j,\alpha,\beta} \geq -C^{\pi s}_{j,\alpha,\beta} & \forall j,\alpha,\beta \\
\eta^{ss}_{j,\alpha,\beta} \geq C^{ss}_{j,\alpha,\beta}, & \eta^{ss}_{j,\alpha,\beta} \geq -C^{ss}_{j,\alpha,\beta} & \forall j,\alpha,\beta \\
\xi \geq 0, \tau \geq 0 & & \\
\eta^s \geq 0, \eta^{\pi s} \geq 0, \eta^{ss} \geq 0 & &
\end{array} \tag{3.14}$$

where f is defined in (3.8).

Theorem 3.1 *The objective functions in programs (3.13) and (3.14) are bounded below by zero.*

Proof: Since the exponential function is always nonnegative and the logarithm function is an increasing function,

$$\log\left(1+e^{f_i}\right) \geq \log(1) = 0 \quad \forall i.$$

Further,

$$\log\left(1+e^{f_i}\right) \geq f_i \quad \forall i.$$

So, since $y_i \in \{0,1\}$,

$$\log\left(1+e^{f_i}\right) \geq y_i f_i \quad \forall i.$$

Thus,

$$\sum_{i=1}^{n}\left[-y_i f_i + \log\left(1+e^{f_i}\right)\right] \geq 0,$$

and, since the penalty terms are nonnegative, the objective functions must be nonnegative. □

Theorem 3.2 *The objective functions in programs (3.13) and (3.14) are convex with respect to the variables in the programs.*

Proof: The penalty terms in the objective functions are linear, as is the y-term. So the only terms we need to consider are the nonlinear logarithm ones.

Let

$$\theta(f) = \sum_{i=1}^{n} \log(1+e^{f_i}).$$

Then

$$\nabla\theta(f) = \left[\frac{e^{f_1}}{1+e^{f_1}}, \cdots, \frac{e^{f_n}}{1+e^{f_n}}\right]^T$$

and

$$\nabla^2\theta(f) = \text{diag}\left(\frac{e^{f_i}}{(1+e^{f_i})^2}\right) \geq 0.$$

So, $\nabla^2\theta$ is positive semi-definite with respect to f, which further implies that θ is convex with respect to f.

Now, the nonlinear terms we are interested in are a composition of a convex ($\theta(f)$) function and linear functions ($f = Td + KC$), so they are themselves convex. Therefore, the objective functions are convex functions. □

By Theorem 3.2 and the fact that all of the constraints are linear, we know that if a local solution exists in program (3.13) or (3.14), then it must be a global solution.

Next, we show that a solution does indeed exist for each of programs (3.13) and (3.14):

Theorem 3.3 *Solutions exist for programs (3.13) and (3.14).*

Proof: Let the objective functions be represented by:

$$g(d, C) = \frac{1}{n}\sum_{i=1}^{n}\left[-y_i f_i + \log\left(1+e^{f_i}\right)\right] + \lambda_\pi^T\|d\|_1 + \lambda_s^T\|C\|_1,$$

where we abuse the notation by letting λ_π and λ_s represent vectors containing the parametric and smooth regularization parameters, respectively. Note that $g(d, C)$ is continuous. Recall from the proof of Theorem 3.1 that

$$\sum_{i=1}^{n}\left[-y_i f_i + \log\left(1+e^{f_i}\right)\right] \geq 0.$$

Thus,

$$g(d, C) \geq \lambda_\pi^T \|d\|_1 + \lambda_s^T \|C\|_1 \tag{3.15}$$

Note that the level set of g

$$L_\nu(g(d, C)) = \{(d, C) | g(d, C) \leq \nu\}$$

is closed since g is continuous. Further, the continuity of g and inequality (3.15) imply that

$$L_\nu(g(d, C)) \subseteq \hat{L}_\nu = \{(d, C) | \lambda_\pi^T \|d\|_1 + \lambda_s^T \|C\|_1 \leq \nu\}. \tag{3.16}$$

Note that $\hat{L}_\nu$ is closed. $\hat{L}_\nu$ is also bounded: since $\lambda_\pi, \lambda_s > 0$ (penalty parameters), we have that

$$\lambda_\pi \|d\|_1 \leq \nu \Rightarrow \|d\|_1 \leq \frac{\nu}{\lambda_\pi}$$

and

$$\lambda_s \|C\|_1 \leq \nu \Rightarrow \|C\|_1 \leq \frac{\nu}{\lambda_s}.$$

So, $\hat{L}_\nu$ is compact. Now, (3.16) implies that $L_\nu(g(d, C))$ is also compact (we already have that it is closed; it is bounded because it is a subset of a larger bounded set, namely $\hat{L}_\nu$).

Since we are minimizing,

$$\min_{(d,C)} g(d, C) = \min_{(d,C) \in L_\nu(g(d,C))} g(d, C).$$

As a result, we want to minimize a continuous function over a compact set. Thus, $g(d, C)$ attains its minimum on $L_\nu(g(d, C))$ [74, p. 198], and thus a solution exists. □

Not only do solutions to programs (3.13) and (3.14) exist, but these solutions are unique.

Theorem 3.4 *The solutions to programs (3.13) and (3.14) are unique.*

Proof: We focus on the additive model (3.13); the proof for the full model (3.14) is almost identical.

Let the objective function be represented by:

$$g(d, C, \xi, \tau) = \frac{1}{n} \sum_{i=1}^{n} \left[-y_i f_i + \log\left(1 + e^{f_i}\right) \right] + \lambda_\pi \xi + \lambda_s \tau.$$

Let $\bar{P} = (\bar{d}, \bar{C}, \bar{\xi}, \bar{\tau})$ be a solution from Theorem 3.3. We want to show that g is strictly convex at this solution.

First, note that since the constraints in model (3.13) are linear, we have a convex constraint set. We consider a scalar $0 < \gamma < 1$ and a feasible point $P = (d, C, \xi, \tau)$, where $P \neq \bar{P}$.

Let $\bar{f} = T\bar{d} + K\bar{C}$ and $f = Td + KC$. We consider the difference:

$$\begin{aligned}
D &= (1-\gamma) g(\bar{P}) + \gamma g(P) - g((1-\gamma)\bar{P} + \gamma P) \\
&= \frac{1}{n} \sum_{i=1}^{n} \big\{ \log[1 + \exp((1-\gamma)\bar{f}_i)] + \log[1 + \exp(\gamma f_i)] - \\
&\quad \log[1 + \exp((1-\gamma)\bar{f}_i + \gamma f_i)] \big\} \\
&= \frac{1}{n} \sum_{i=1}^{n} \log \left\{ \frac{[1 + \exp((1-\gamma)\bar{f}_i)][1 + \exp(\gamma f_i)]}{1 + \exp((1-\gamma)\bar{f}_i + \gamma f_i)} \right\} \\
&= \frac{1}{n} \sum_{i=1}^{n} \log \left[\frac{1 + \exp((1-\gamma)\bar{f}_i + \gamma f_i) + \exp((1-\gamma)\bar{f}_i) + \exp(\gamma f_i)}{1 + \exp((1-\gamma)\bar{f}_i + \gamma f_i)} \right] \\
&> \log(1) = 0
\end{aligned}$$

since $\exp((1-\gamma)\bar{f}_i) > 0$ and $\exp(\gamma f_i) > 0$. Therefore, g is strictly convex by definition.

Applying the Uniqueness Theorem from [74, p. 73], we have that $\bar{P}$ is the unique solution to the program. □

3.2 Fitting the Models

Given values for the regularization parameters, programs (3.13) and (3.14) return corresponding values for the d and C variables, and thus define f (and p). So in order to find p, we first need to choose values for the regularization parameters that give the smallest misclassification rate.

With a slight abuse of notation, we let λ represent the set of regularization parameters for each model. In other words, for the additive model (3.13), $\lambda = (\lambda_\pi, \lambda_s)$; while for the full model (3.14), $\lambda = (\lambda_\pi, \lambda_{\pi\pi}, \lambda_s, \lambda_{\pi s}, \lambda_{ss})$.

3.2.1 Parameter Selection: CKL and GACV

Let p be the true, unknown probability function from which y is determined, and let p_λ be the corresponding estimate associated with λ. Similarly, let f be the true log odds ratio, while f_λ is the corresponding estimate. Note that because we have Bernoulli data ($y_i = 0, 1$), the mean of y_i is $\mu_i = p_i$ and the variance is $\sigma_i^2 = p_i(1-p_i)$ [117]; let $\mu_{\lambda i}$ and $\sigma_{\lambda i}^2$ represent the corresponding λ estimates. Finally, let $b(f) = \log(1+e^f)$ (this definition

comes from the fact that we are dealing with an exponential function; see for example, [66, 106, 112]).

The Kullback-Leibler (KL) distance is often used to measure the distance between two probability distributions. In our case (Bernoulli data) [66, 118],

$$KL(p, p_\lambda) = \frac{1}{2}[\mu(f - f_\lambda) - (b(f) - b(f_\lambda))].$$

Considering only the values that depend upon λ in KL, we obtain the comparative Kullback-Leibler (CKL) distance [118]:

$$CKL(p, p_\lambda) = \frac{1}{n}\sum_{i=1}^{n}[-\mu_i f_\lambda(x_i) + b(f_\lambda(x^i)].$$

CKL is an upper bound for the expected misclassification rate [105].

In order to calculate CKL, $\mu = p$ must be known. p is known for simulated data but not for real data. Thus, we need a proxy for CKL to use on real data. As a proxy, we use the generalized approximate cross validation (GACV) distance, described in [66, 67, 105].

As Lin et al. [66] note, it is tempting to estimate CKL by

$$OBS(\lambda) = \frac{1}{n}\sum_{i=1}^{n}[-y_i f_\lambda(x_i) + b(f_\lambda(x_i))],$$

particularly since OBS (for "observed") is easily obtained from a solution to the program (3.13) or (3.14). However, OBS is known to be an underestimate of CKL (y_i and $f_\lambda(x_i)$ are correlated). Thus,

$$CKL(\lambda) = OBS(\lambda) + D(\lambda)$$

for some D. It is D that we would like to estimate.

Define

$$CV(\lambda) = \frac{1}{n}\sum_{i=1}^{n}[-y_i f_\lambda^{[-i]}(x_i) + b(f_\lambda(x_i))],$$

where $f_\lambda^{[-i]}$ is the solution to program (3.13) or (3.14) for a particular λ with observation i left out (CV is the "leave-one-out" cross-validation function). Then CV is a proxy for CKL [66, 105]. (Note that y_i and $f_\lambda^{[-i]}(x_i)$ are uncorrelated.) However, computing CV would require n solutions to the program for every value of λ. So, instead of calculating CV directly, we approximate it.

Now,

$$CV(\lambda) = OBS(\lambda) + \frac{1}{n}\sum_{i=1}^{n} y_i(f_\lambda(x_i) - f_\lambda^{[-i]}(x_i)).$$

After a series of approximations (whose details can be found in [116]), we obtain the approximate cross validation (ACV), which is a second-order approximation to CV:

$$ACV = OBS(\lambda) + \frac{1}{n}\sum_{i=1}^{n} h_{ii}\frac{y_i(y_i - \mu_\lambda(x_i))}{1 - \sigma_{\lambda i}^2 h_{ii}}$$

where

$$h_{ii} \approx \frac{f_\lambda(x_i) - f_\lambda^{[-i]}(x_i)}{y_i - \mu_\lambda^{[-i]}(x_i)}$$

(see [116] for more). Letting $W = \mathrm{diag}(\sigma_{\lambda i}^2)$ and replacing h_{ii} with $\frac{1}{n}\sum_{i=1}^{n} h_{ii} = \frac{1}{n}\mathrm{tr}(H)$, we finally obtain

$$GACV(\lambda) = \frac{1}{n}\sum_{i=1}^{n}[-y_i f_\lambda(x_i) + b(f_\lambda(x_i))] + \frac{\mathrm{tr}(H)}{n}\frac{\sum_{i=1}^{n} y_i(y_i - \mu_\lambda(x_i))}{\mathrm{tr}[I - W^{1/2}HW^{1/2}]}.$$

Note that the calculation of GACV depends upon the inversion of a large-scale matrix. To make this easier to compute, we follow [118] by defining the randomized GACV (ranGACV), a computable proxy for GACV. Let $\epsilon = (\epsilon_1, \ldots, \epsilon_n)^T$ be a zero-mean random vector of independent components with variance σ_ϵ^2. Let f_λ^y be the minimizer of the program (3.13) or (3.14) using the original data y. Let $f_\lambda^{y+\epsilon}$ be the minimizer using the perturbed data $y + \epsilon$. Then, from [118],

$$\begin{aligned} ranGACV &= \frac{1}{n}\sum_{i=1}^{n}\left[-y_i f_\lambda(x_i) + \log(1 + e^{f_\lambda(x_i)})\right] \\ &\quad + \frac{\epsilon^T(f_\lambda^{y+\epsilon} - f_\lambda^y)}{n}\frac{\sum_{i=1}^{n} y_i(y_i - \mu_\lambda(x_i))}{\epsilon^T\epsilon - \epsilon^T W(f_\lambda^{y+\epsilon} - f_\lambda^y)}. \end{aligned} \tag{3.17}$$

Thus, to calculate ranGACV, we only need the solution to two problems: the original one with y and the perturbed one with $y + \epsilon$.

Unlike the original problem, which is bounded below by zero as shown in Theorem (3.1), the perturbed problem may not be bounded below. Consider the case where $y_i = 1$ and $\epsilon_i > 0$ (so that $y_i + \epsilon_i > 1$). The i-th term of the sum in $-\log(L)$ becomes:

$$-(1 + \epsilon_i)f_i + \log\left(1 + e^{f_i}\right) \stackrel{f_i \to \infty}{\longrightarrow} -\epsilon_i f_i \to -\infty.$$

If the penalty terms are not large enough (depending upon T and K) to cancel $-\epsilon_i f_i$, then it is possible for the perturbed problem to become unbounded. This has happened in our tests on simulated data. In such cases, we need to decrease ϵ_i.

3.2.2 Choice of Solver

The programs (3.13) and (3.14) are not difficult programs to solve, as all of the constraints are linear and the programs are convex. However, finding ranGACV for real instances can be difficult due to the size of the data and the number of individual solves that must be performed.

The size of individual programs depends upon the number of observations (n), the number of dimensions for the observations (D), and the number of dimensions for the basis of the kernel (N). For testing purposes, we use simulated data where $n = 1000, D = 10$ and $N = 50$. Since both matrices T and K in the calculation of f (from equation (3.9)) are dense, this results in a large number of values that must be stored to calculate f: for the additive model, T is 1000×11 and K is 1000×50; for the full model, T is 1000×56 and K is 1000×7250 (requiring approximately 58 MB of storage). Real-life data is typically even larger [115]: n often varies between 2000 and 4000, with 10000 being considered a large data set; D often varies between 20 and 50; and N is around $n/20$. Because of this, we would like to avoid storing T and K — in fact, in the full model, we often cannot store K entirely.

The data that is stored also affects the number of variables and constraints in the programs. Not storing K entirely in the full model means that the f variables cannot be explicitly defined in the model, but must be implicitly used. This reduces the number of variables and constraints by n, but makes the definition of the objective function more complicated. We've moved the

issue from one of storage to one of model formulation.

To find ranGACV, program (3.13) or (3.14) must be solved twice for every grid point — once for the original problem and once for the perturbed problem. The number of times this must be done depends upon how we choose λ.

These requirements mean that we must choose a solver that does not require an explicit definition for f. In addition, we would like one that is fast and consistent since we are potentially doing many solves (depending on the choices of λ). We use MINOS [82] as the underlying solver. This choice is based upon tests performed under GAMS using the original additive model on simulated data. Two GAMS models were written, one using explicit f-variables and one using only the d, C variables. Both models were submitted to GAMS for various λ under five different nonlinear solvers: CONOPT, CONOPT2, MINOS, MINOS5, and SNOPT. Refer to Tables 4 and 5 for the results. Only CONOPT and MINOS returned the same objective values for both GAMS models, suggesting only these two solvers were consistent on our problems. Further, MINOS's results were smaller than CONOPT's, suggesting that MINOS found a better point.

The last two rows in Tables 4 and 5 give the total solution times (resource usage) and the total overall times spent in GAMS. These results strongly suggest that using explicit f-variables improves solution efficiency. However as noted above, including explicit f variables increases the total number of

λ values	CONOPT	CONOPT2	MINOS	MINOS5	SNOPT
$\lambda_\pi = 2^{-20}$ $\lambda_s = 2^{-20}$	0.541399	0.540633	0.540831	0.540616	0.540905
$\lambda_\pi = 2^{-20}$ $\lambda_s = 2^{-19}$	0.541441	0.540927	0.540995	0.540924	0.540938
$\lambda_\pi = 2^{-19}$ $\lambda_s = 2^{-20}$	0.541404	0.540630	0.540760	0.540620	0.540784
$\lambda_\pi = 2^{-19}$ $\lambda_s = 2^{-19}$	0.541446	0.540929	0.540940	0.540928	0.540943
Solve Time	9.292	58.967	16.214	15.680	19.275
Total Time	27.14	77.08	34.60	33.97	37.49

Table 4: Objective values obtained by solving the original additive model with explicit f-variables.

λ values	CONOPT	CONOPT2	MINOS	MINOS5	SNOPT
$\lambda_\pi = 2^{-20}$ $\lambda_s = 2^{-20}$	0.541399	0.540620	0.540831	0.540831	0.540905
$\lambda_\pi = 2^{-20}$ $\lambda_s = 2^{-19}$	0.541441	0.540925	0.540995	0.540995	0.540940
$\lambda_\pi = 2^{-19}$ $\lambda_s = 2^{-20}$	0.541404	0.540628	0.540760	0.540836	0.540782
$\lambda_\pi = 2^{-19}$ $\lambda_s = 2^{-19}$	0.541446	overflow	0.540940	0.541000	0.540945
Solve Time	22.047	164.560	124.358	73.530	99.537
Total Time	486.87	628.90	589.82	538.65	565.38

Table 5: Objective values obtained by solving the original additive model without explicit f variables.

variables and the total number of constraints (both by n), and requires that K be stored in its entirety. This causes difficulty primarily for the full model, where often K cannot be fully stored and only K_1 is stored. Further, many of the gains in solution time seen in Table 4 as compared to Table 5 result from eliminating iterations that generate nonsensical values for f. To obtain the values in Table 4, we restricted the f-values to be less than or equal to 30 in absolute value. However, this was not possible for those results in Table 5 since f was not explicitly present in the model. However, when using the solvers directly an intermediate comparison of the f-value can be made and infinity can be returned for values outside of reasonable ranges. Thus, we develop models that do not use explicit f variables.

3.2.3 Parameter Search: Grid Search

Our goal is to choose the λ values that minimize the ranGACV measure. For each set of λ's, we need to solve two nonlinear programming problems (an original and a perturbed problem) in order to determine the corresponding value for ranGACV. Thus, we have a nested set of minimization problems, where the inner minimization (solving model (3.13) or (3.14)) is a straight nonlinear problem, and the outer minimization (minimizing ranGACV) makes use of these results. Note that the outer minimization is almost constraint-free: the only constraint is that $\lambda > 0$ since λ represents weight on penalty parameters.

The simplest procedure to find ranGACV is to divide the λ-space into a grid, and calculate ranGACV at each grid point. Once the calculations are complete, we can choose the λ values by selecting the grid point for which ranGACV achieved a minimum. Essentially, we change the problem to a minimization over a finite set. Because of this, we may not find the true minimum. However, this procedure is always implementable although costly depending upon the grid size. We use a grid of varying step size: we move from one set of λ values to the next by dividing the λ values in half. This provides a coarse grid for large values of λ (where the focus of the model is on minimizing the penalty parameters), and a fine grid for small values of λ (where the focus is on maximizing the likelihood). We consider λ ranging from 2^{-1} to 2^{-20} [115].

Under the grid search, we may have hundreds or thousands of individual solves, depending upon the range for and number of λ. In order to obtain solutions in a reasonable amount of time, we need to employ an efficient solving approach. Note that like the problems presented in Chapter 2, we can consider the values of λ and the choice of y to be individual slices of data, as only these values change between solves. Thus LBP can be reduced to an example of nonlinear slice modeling. We have a series of programs — one for each type (original or perturbed) and grid point — that only differ in the data used to define them (λ,y). By applying slice modeling ideas to the LBP programs, we can improve efficiency and make the grid search usable.

Under MINOS, nonlinear programs are specified in three pieces: the linear portion, the nonlinear objective function, and the nonlinear constraints. Originally, MINOS required the linear portion of the program to be specified by an MPS file; later versions of MINOS include the subroutine `minoss`, that reads the linear portion from parameters. Using `minoss`, we are able to specify and store the linear portion of the LBP programs internally, eliminating the need to write a new MPS file every time we move to a new grid point (that is, change the λ's). Besides saving us time in accessing files, it also enables us to hold the program structure constant throughout all solves.

The nonlinear objective function is specified by the subroutine `funobj`. Besides including the nonlinear objective portion, we also include all f-related terms in `funobj` since we do not model f explicitly. Not only does this provide for easy changes between the original and perturbed solves (simply specify which y values to use), this also simplifies the f-calculations: we only need to calculate f inside of `funobj`, rather than in the linear portion as well.

We can speed up individual solves in MINOS by using a hot start. On the first solve, we use a cold start. The cold start instructs MINOS to use a crash procedure to determine an initial basis [82]. All subsequent solves use a hot start. Under the hot start, not only is the basis from the previous call maintained (as in a warm start), but so are the LU factors of the basis, the approximate reduced Hessian, and the column and row scales [82].

Problem	Original	Slice Techniques	Separate Solves
$n = 500$ $D = 5$ $N = 25$	1.5 hours	12 sec	6 sec
$n = 1000$ $D = 10$ $N = 50$	23 hours	4 min	40 sec
$n = 2000$ $D = 20$ $N = 100$	72+ hours	17 min	7 min

Table 6: Time comparison on additive problems.

This is particularly advantageous for the models (3.13) and (3.14) since the constraint set does not change between between solves. This significantly speeds up subsequent individual solves.

Table 6 compares the times for solving the additive model (3.13) on three simulated problems of increasing size. For all problems, we let λ_π and λ_s range from 2^{-1} to 2^{-5}, for a total of 25 solves. The first column (Original) displays the times for solving the model using code from [115]. This code uses the subroutine `minos1`, which requires an MPS file to written for each program. In addition, for each set of λ values, it solves the original and perturbed versions together followed by the ranGACV calculation. Note that for the largest problem, we never actually completed all 25 solves.

The second column (Slice Techniques) displays the results for applying the slice modeling approach to LBP. In this code, we use the subroutine `minoss`. This eliminates the need for MPS files, allowing us to store the programs internally and maintain program structure between solves. In addition, we move all f-related terms to `funobj` and we request hot starts for all but the first solve. Like the original code, this version solves the original program, followed immediately by the perturbed program for each set of λ values. As can be seen from Table 6, moving to an internal problem representation gives a significant time improvement, requiring only minutes instead of hours to solve all 25 programs.

We note that we can further improve efficiency by considering the order of the solves. Once we have the solutions for the original and perturbed problems at a particular grid point, ranGACV can be calculated. This suggests the approach of solving the original and perturbed problems together for each grid point, as we have done. However, the slice modeling approach suggests the opposite: because fewer changes take place moving from one grid point to another while maintaining the problem type, previous solutions will have greater impact on future solves if the original and perturbed solves are separated. Such separation requires extra storage: we must store the d and C variable values for each solve so that ranGACV can be calculated later. If these solution files become too large, multiple solves must be done, each time using only a piece of the grid. But we do improve the solution

time, as can be seen in Table 6. This also allows for much easier parallel implementation — now the original and perturbed problem can be run in parallel.

The final column (Separate Solves) displays the times for ordering the solves according to the slice modeling approach. The only difference between this approach and that of the second (Slice Techniques) approach is that the original and perturbed programs are solved separately. (The times displayed include the times to calculate the results, after all solves have finished.) Solving the problems separately reduces the improved times by half or more. This time savings enabled us to solve more complicated models, such as the full model (3.14) or the categorical models discussed in [118].

3.3 Alternative Parameter Selection Methods

Using the simple grid search for the parameter selection, we are guaranteed to find a minimum over the set of λ's that we consider. Considering a different set of λ's may result in a better or worse ranGACV value. Thus, we must consider a large set of λ's in order to guarantee a relatively good solution. The approach we use above, varying the size of the steps between grid points, allows us to focus more on areas where ranGACV is small. When λ is large (near 2^{-1}), the penalty terms dwarf the likelihood terms, and the d, C variables are set equal to zero in the solution (the only nonzero variable

is the model's constant term, d_0, which is not penalized). Thus, we end up ignoring the likelihood calculation completely and so ranGACV is likely to be large (since maximum likelihood is small). When λ is small, the penalty parameters are small enough that likelihood is considered and so ranGACV decreases. By using a finer grid for smaller λ, we are more likely to obtain a small value of ranGACV. Although this grid search technique reduces the number of ranGACV calculations we must do, we are still not guaranteed a minimum ranGACV — another set of λ's may produce a smaller ranGACV.

Further under a grid search, we consider all combinations of λ values. We cannot adapt the search to concentrate on areas where ranGACV is likely to be small because we have no mechanism for determining such areas. This results in a number of wasted calculations. To reduce the overall solution time, we would like to use an efficient and fast search method to determine the λ values that give the minimum ranGACV. In other words, we would like an alternative method for performing the outer minimization.

In this section, we consider three derivative-free search methods: Nelder-Mead Simplex method [61] as implemented in the routine `fminsearch` in MATLAB [80]; Powell's UOBYQA method [88], and the Wedge Trust Region method [79]. All three methods minimize an unconstrained function of several variables. Under these methods, the λ values are the variables and the ranGACV calculation is the objective function. Because these methods are derivative-free, we do not need an explicit functional formulation for the

objective function; rather, we can specify the objective function as a subroutine. Input to our objective function subroutine are the λ values; output is the corresponding ranGACV value. Contained within this subroutine are two nonlinear MINOS solves, one for the original model and one for the perturbed model.

In the following, the objective function (ranGACV) is denoted by F and the variables (λ) are denoted by the vector x. For example for the additive model, $\lambda_\pi = x_1$ and $\lambda_s = x_2$. The number of variables is denoted by n. For the additive model, $n = 2$.

3.3.1 Nelder-Mead Simplex Method

The Nelder-Mead Simplex method is a direct search method: it attempts to minimize a function of n real variables using only function values, not derivative information. To do this the method maintains a simplex, the convex hull of $n + 1$ distinct points (vertices). Here, we present the main algorithm from the MATLAB procedure `fminsearch` [80], which is based upon the algorithm presented in [61].

Algorithm 3.1 *`fminsearch` Method*

1. *If the diameter of the simplex is less than a given tolerance ($tolx$)* and *the function values differ from the minimum by less than a given tolerance ($tolf$) or the maximum function evaluations have been exceeded,*

terminate. Otherwise, continue.

2. *Order the vertices of the simplex according to their corresponding function values, from smallest to largest. (We look for a point to replace the worst $(n+1)$ vertex.)*

3. *Build the reflection point, x_r, from the centroid of the n best vertices and the worst $(n+1)$ vertex:*

$$x_r = (1+\rho)\bar{x} - \rho x_{n+1}.$$

(In `fminsearch`*, $\rho = 1$.)*

4. *(Expansion) If $F(x_r) < F(x_1)$, then calculate the expansion point x_e:*

$$x_e = (1+\rho\chi)\bar{x} - \rho\chi x_{n+1}.$$

(In `fminsearch`*, $\chi = 2$.) If $F(x_e) < F(x_r)$, set $x_{n+1} = x_e$; otherwise, set $x_{n+1} = x_r$. Terminate the current iteration and return to Step 1.*

5. *(Reflection) If $F(x_r) < F(x_n)$, set $x_{n+1} = x_r$. Terminate the current iteration and return to Step 1.*

6. *(Contraction) We have that $F(x_r) \geq F(x_n)$, and we perform a contraction between $\bar{x}$ and the better of x_{n+1} and x_r.*

   - *If $F(x_r) < F(x_{n+1})$, perform an outside contraction. Calculate*

$$x_c = (1+\psi\rho)\bar{x} - \psi\rho x_{n+1}.$$

(In **fminsearch**, $\psi = 0.5$.*) If* $F(x_c) \leq F(x_r)$, *set* $x_{n+1} = x_c$, *terminate the current iteration and return to Step 1. Otherwise, go to Step 7.*

- *We have that* $F(x_r) \geq F(x_{n+1})$, *and we perform an inside contraction. Calculate*

$$x_{cc} = (1 - \psi)\bar{x} + \psi x_{n+1}.$$

If $F(x_{cc}) < F(x_{n+1})$, *set* $x_{n+1} = x_{cc}$, *terminate the current iteration and return to Step 1. Otherwise, go to Step 7.*

7. *(Shrink Step) We shrink the simplex. Set*

$$v_i = x_1 + \sigma(x_i - x_1)$$

for $i = 2, \ldots, n + 1$. *(In* **fminsearch**, $\sigma = 0.5$.*) Set the vertices of the simplex to be* $x_1, v_2, \ldots, v_{n+1}$. *Terminate the current iteration and return to Step 1.*

In `fminsearch` the user can specify the tolerances $tolf$ and $tolx$, the maximum number of function evaluations and the maximum number iterations.

3.3.2 Powell's UOBYQA Method

UOBYQA stands for *U*nconstrained *O*ptimization *BY Q*uadratic *A*pproximation. It forms quadratic models for the function F by interpolating $m = (n+1)(n+$

2)/2 values [88]. Usually, an iteration consists of finding a new vector of variable values by either the minimization the quadratic model subject to a trust region bound (a trust region step), or a procedure that is used to improve the accuracy of the quadratic model (a model step). We summarize the algorithm from [88]:

Algorithm 3.2 *UOBYQA Method*

1. *(Initialization) The user specifies the initial vector of variables x^b, the beginning trust region radius ρ_{beg}, and the ending trust region radius ρ_{end}. From the user-specified information, the initial set of interpolation points x^i are generated. The quadratic model Q is defined from these points:*

$$Q(x) = c_Q + g_Q^T(x - x^b) + \frac{1}{2}(x - x^b)^T G_Q(x - x^b)$$

where the scalar c_Q, the vector g_Q, and the $n \times n$ symmetric matrix G_Q can be found from the interpolation equations

$$Q(x^i) = F(x^i).$$

Note that the j-th Lagrange function of the interpolation problem is given by [88]:

$$\ell_j(x) = c_j + g_j^T(x - x^b) + \frac{1}{2}(x - x^b)^T G_j(x - x^b),$$

which implies that

$$c_Q = \sum_{j=1}^{m} F(x^j)c_j,$$

$$g_Q = \sum_{j=1}^{m} F(x^j) g_j,$$

and

$$G_Q = \sum_{j=1}^{m} F(x^j) G_j.$$

Set the trust region radius ρ *to* $\rho = \rho_{beg}$*, and another radius* Δ *to* $\Delta = \rho_{beg}$*. (*$\Delta \geq \rho$ *allows the lengths of the changes to the variables to exceed* ρ*, thus avoiding some loss of efficiency since the algorithm never actually increases* ρ*.) Identify* x^k *as the best of the interpolation points:*

$$F(x^k) = \min\{F(x^i) | i = 1, \ldots, m\}.$$

Finally, set $j = 0$*.* j *determines whether we have a trust region step (*$j = 0$*) or a model step (*$j > 0$*); when we have a model step,* j *specifies the index of the interpolation point to update.*

2. *(Trust Region Step) If* $j = 0$*, then solve*

$$\min_d Q(x^k + d) \text{ subject to } \|d\|_2 \leq \Delta$$

for the trial step d*. If* $\|d\|_2 < \rho/2$*, then go to Step 6.*

3. *(Update) We do two updates:*

   - *If* $j = 0$*, update the radius* Δ *according to standard trust region techniques. However, if the updated* Δ *is close enough to* ρ *(*$\Delta \leq \frac{3}{2}\rho$*), then set* $\Delta = \rho$ *(occasionally, it is helpful not to let* Δ *exceed* ρ *[88]).*

- *Identify the interpolation point* x^t *to be replaced by* x^k+d *if* $F(x^k+d) < F(x^k)$. *If* $j > 0$, $t = j$; *otherwise, look for an interpolation point far from the best current point. If no replacement is found,* $t = 0$.

4. *(Move) If* $t > 0$, *then set* $x^t = x^k + d$, *and update the quadratic model.*

5. *(Test) Test to determine if the current quadratic model is adequate. If an interpolation point has been moved* ($t > 0$), *this is done by considering whether the move resulted from a model step* ($j > 0$); *whether the move resulted in a better minimum function value; whether the move was large enough* ($> 2\rho$); *or whether the trust region* Δ *is large enough* ($> 2\rho$). *If the model is found to be adequate, then set* $j = 0$, *terminate the current iteration, and return to Step 2.*

6. *(Model Step) Improve the accuracy of the quadratic model. The general idea is to move the "worst" point* x^j *by* d *(in a general sense,* x^j *is farthest point from* x^k*).* d *is determined by maximizing the Lagrange function for the interpolation point* j*:*

$$\max_d |\ell_j(x^k + d)| \quad \textit{subject to } \|d\|_2 \leq \rho.$$

If no "worst" point x^j *can be found, then set* $j = 0$.

7. *If* $j = 0$, *or if* $j > 0$ *and* $\|d\|_2 > \rho$, *terminate the current iteration and go to Step 2. (We've improved the accuracy of the model if necessary*

by the model step, and now we concentrate on the trust region step.)

8. *If $\rho > \rho_{end}$, then decrease ρ and Δ, and revise $x^b = x^b + x^k$. Terminate the current iteration and go to Step 2.*

9. *Finish any updates and return the current x^k to the user as the best estimate of the optimal vector.*

For implementation, we use the Fortran code provided by Powell [87].

3.3.3 Wedge Trust Region Method

In a similar fashion to the UOBYQA method, the Wedge Trust Region method generates a model (either linear or quadratic) for F by interpolating the function at a series of points. The difference lies in the trust region problem. To ensure that the model is well-defined, the geometric condition that the interpolation points are linearly independent (non-degenerate) must be satisfied. Other methods (like Powell's UOBYQA) include a model step that improves the accuracy of the model if the geometric condition fails. Marazzi and Nocedal [79] instead incorporate the geometric condition explicitly in the trust region step. When the model is linear, this constraint takes on the form of a wedge (hence the name). Here, we summarize the algorithm from [79]:

Algorithm 3.3 *Wedge Algorithm*

1. *(Initialization) Choose the trust region parameters* $\alpha, \beta \in (0,1)$ *(*α, β *control how fast the trust region grows or shrinks). Let the initial trust region radius be* $\Delta_c > 0$ *and the initial guess be* x^c*. The interpolation points* $\Sigma_c = \{y^i | i = 1, \ldots, m\}$ *(*$m = n$ *for a linear model and* $m = (n+1)(n+2)/2 - 1$ *for a quadratic model) are selected so that* $\{x^c\} \cup \Sigma_c$ *is non-degenerate. Assume that the current iterate is the best of the interpolation points:* $F(x^c) \leq F(y^i)$*.*

2. *Find the point farthest from the current point:*

$$y^{out} = \arg\max_{y \in \Sigma_c} \|y - x^c\|_2.$$

3. *Form the (linear or quadratic) model,* m_c*, that interpolates* $x^c \cup \Sigma_c$*:*

$$m_c(x_c + s) = F(x^c) + g_c^T s + \frac{1}{2} s^T G_c s,$$

where the vector g_c *and the* $n \times n$ *symmetric matrix* G_c *are determined from the interpolation points (*$G_c \equiv 0$ *for the linear model). Define the wedge* $\mathcal{W}_c$*.* $\mathcal{W}_c$ *contains the "taboo region" of all points* $x_c + s$ *that would result in a degenerate set of interpolation points, as well as points near the "taboo region" (this keeps the system from being too ill-conditioned [79]).*

4. *Compute the step s_c by approximately solving:*

$$\begin{aligned} \min_{s} \quad & m_c(x_c + s) = F(x^c) + g_c^T s + \tfrac{1}{2} s^T G_c s \\ \textit{subject to} \quad & \|s\|_2 \leq \Delta_c \\ & s \notin \mathcal{W}_c. \end{aligned}$$

For the linear model, the wedge constraint requires that the magnitude of the cosine of the angle between the step s and the normal b_c be larger than a given constant $\gamma \in (0, 1)$ and so the last constraint becomes:

$$|b_c^T s| \geq \gamma \|b_c\|_2 \|s\|_2.$$

For the quadratic model, this generalizes to

$$|b_c^T s + \frac{1}{2} s^T B_c s| \geq \gamma \|s\|_2 \sqrt{1 + \frac{1}{2}\|s\|_2^2}.$$

Here, B_c is the $n \times n$ symmetric matrix that generalizes the normal b_c from the linear case; B_c can be obtained from a QR factorization [79].

5. *Set ared$(s_c) = F(x_c) - F(x_c + s_c)$ and pred$(x_c) = m_c(x_c) - m_c(x_c + s_c)$.*

6. *Update the trust region radius Δ_c: if ared$(s_c) >$ αpred(s_c), choose Δ_+ such that $\Delta_+ \geq \Delta_c$. Otherwise, set $\Delta_+ = \beta \Delta_c$.*

7. *If $F(x_c + s_c) < F(x_c)$ (successful iteration),*

   - *update the current iterate: $x_+ = x_c + s_c$, and*

- *replace* y^{out} *with the (old) iterate* x_c*:* $\Sigma_+ = \{x_c\} \cup \Sigma_c \backslash \{y^{out}\}$.

Otherwise (unsuccessful iteration),

- *Maintain the current iterate:* $x_+ = x_c$*, and*
- *If the new trial point* $x_c + s_c$ *is not farther from* x_c *than* y^{out}*, replace* y^{out}*; otherwise, discard the new trial point:*

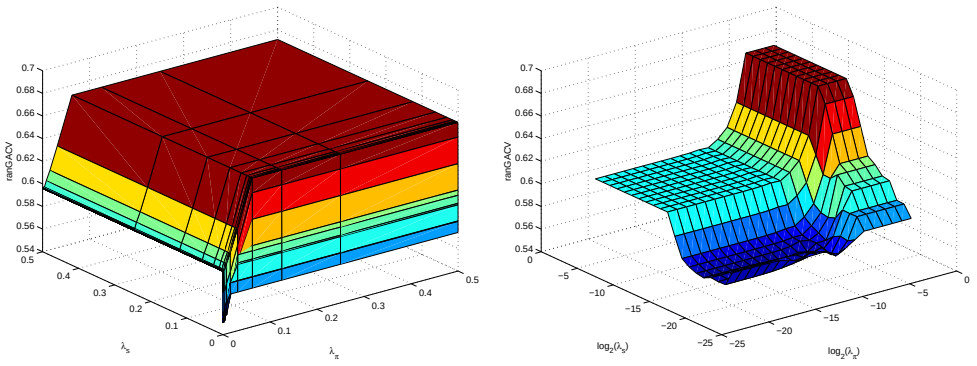
$$\Sigma_+ = \begin{cases} \{x_c + s_c\} \cup \Sigma_c \backslash \{y^{out}\} & \text{if } \|y^{out} - x_c\|_2 \geq \|(x_c + s_c) - x_c\|_2 \\ \Sigma_c & \text{otherwise.} \end{cases}$$

8. *Set* $x_c = x_+$*,* $\Sigma_c = \Sigma_+$*, and* $\Delta_c = \Delta_+$*. Terminate the current iteration and return to Step 2.*

To implement this algorithm, we use the MATLAB code for the quadratic model provided by Marazzi [78], with (default values) $\alpha = 0.25$, $\beta = 0.75$, and $\gamma = 0.4$.

3.3.4 Results

We compare the three derivative-free search methods to each other and to grid searches on three problems under the additive model (3.13). The first problem uses simulated data with $n = 1000$, $D = 10$, and $N = 50$. Figure 9 shows ranGACV from a 20×20 grid search ($\lambda = 2^{-1}, 2^{-2}, 2^{-3}, \ldots, 2^{-20}$). Displayed are plots for both the standard and log-log (base 2) scales. The



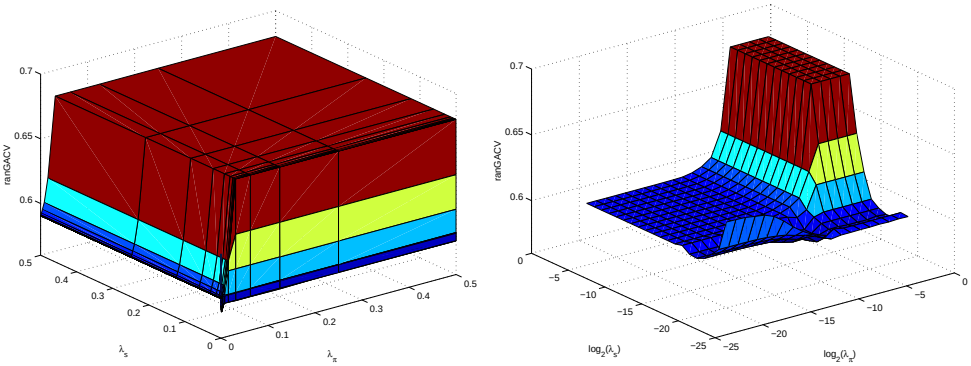
(a) Standard Scale (b) Log-Log Scale

Figure 9: ranGACV for the simulated data problem.

other two problems use data drawn from medical applications. The first has $n = 669$, $D = 6$, and $N = 40$. The second has $n = 668$, $D = 6$, and $N = 40$. Figures 10 and 11 show ranGACV from the 20×20 grid search on these two (real) problems under both the standard and log-log (base 2) scales.

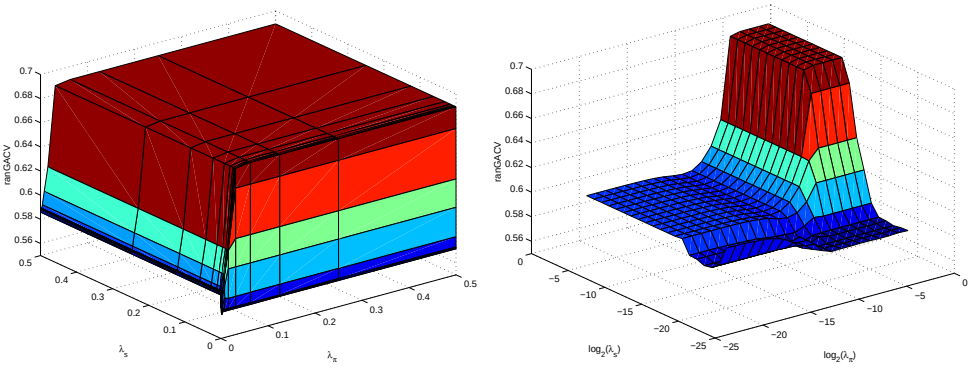
As can be seen from the three figures on the standard scale, ranGACV is a difficult function to minimize: it is a flat plateau at large λ values, but drops sharply at small λ values. The log-log (base 2) plots show a nicer function. The flat plateau at large λ values is smaller. Another flat plateau occurs at a smaller ranGACV value, but this feeds into a ridge containing the minimum as λ_s decreases. For this reason, we do a change of variables to make the log-log space our search space:

$$x_1 = \log_2(\lambda_\pi) \text{ and } x_2 = \log_2(\lambda_s).$$



(a) Standard Scale (b) Log-Log Scale

Figure 10: ranGACV for the first real data problem.



(a) Standard Scale (b) Log-Log Scale

Figure 11: ranGACV for the second real data problem.

Note that due to this change of variables, we do not have to worry about maintaining $\lambda > 0$: since we apply the algorithms in the log-log (base 2) space, all corresponding λ values will be $2^x > 0$.

To obtain appropriate starting points for the derivative-free methods, we consider multiple grid searches. We start with a coarse 5×5 grid: $x_1, x_2 \in \{-4, -8, -12, -16, -20\}$. Then, we try a finer 10×10 grid: $x_1, x_2 \in \{-2, -4, -6, \ldots, -18, -20\}$. Finally, we consider the typical 20×20 grid: $x_1, x_2 \in \{-1, -2, -3, \ldots, -19, -20\}$. We start the derivative-free methods at the best grid point found under each grid search.

Initial trust region radii for the UOBYQA and Wedge methods are set large for the coarse grid and are decreased on the finer grids: for the 5×5 grid, the radii are set to 4; for the 10×10 grid, the radii are set to 2; and for the 20×20 grid, the radii are set to 1. This is done because we expect the grid search to come closer to the minimum as the number of grid points is increased.

All three search methods are allowed to run until the tolerances were satisfied, so any maximum iteration and maximum function evaluation limits are set large enough to be ignored. For the Nelder-Mead simplex method, we set $tolx = tolf = 1 \times 10^{-8}$. For the trust region methods, the terminating trust region radii are also set to 1×10^{-8}.

Tables 7, 8, and 9 display the ranGACV value obtained and the number of function evaluations under each search method, starting at the specified

	5×5 Grid	10×10 Grid	20×20 Grid
Best Grid Search Pt	$(-12, -16)$	$(-12, -16)$	$(-12, -16)$
Grid Search	0.55029982 (25 func evals)	0.55029982 (100 func evals)	0.55029982 (400 func evals)
Nelder-Mead	0.54981480 (122 func evals)	0.54981480 (122 func evals)	0.54981480 (122 func evals)
UOBYQA	0.54978822 (86 func evals)	0.54978498 (85 func evals)	0.54982361 (77 func evals)
Wedge	0.54979829 (45 func evals)	0.54986966 (38 func evals)	*0.54968777* (40 func evals)

Table 7: Search method results for the simulated data problem.

starting point. The best ranGACV value is emphasized in each table.

In all cases, the derivative-free search methods improved upon the corresponding grid search value. In almost all cases (except for UOBYQA on the coarse grid in the second real problem), the methods also improved upon *all* of the grid searches. This suggests that even a coarse grid search is sufficient for obtaining appropriate starting points. (Note that starting the methods cold from no grid search point resulted in worse solutions, so some initial search must be done to obtain a good starting point.) Further, the derivative-free search methods on coarser grids required significantly fewer total function evaluations then the 400 needed for the finest 20×20 grid search. The maximum total function evaluations on the 5×5 grid was

	5×5 Grid	10×10 Grid	20×20 Grid
Best Grid Search Pt	$(-8, -16)$	$(-10, -16)$	$(-10, -16)$
Grid Search	0.58289425	0.58005256	0.58005256
	(25 func evals)	(100 func evals)	(400 func evals)
Nelder-Mead	*0.57862580*	0.57934657	0.57934657
	(112 func evals)	(80 func evals)	(80 func evals)
UOBYQA	0.57872897	0.57936501	0.57899692
	(82 func evals)	(88 func evals)	(111 func evals)
Wedge	0.57931261	0.57871935	0.57883227
	(68 func evals)	(56 func evals)	(54 func evals)

Table 8: Search method results for the first real data problem.

	5×5 Grid	10×10 Grid	20×20 Grid
Best Grid Search Pt	$(-8, -16)$	$(-10, -16)$	$(-9, -16)$
Grid Search	0.57265086	0.57103974	0.56970404
	(25 func evals)	(100 func evals)	(400 func evals)
Nelder-Mead	*0.56792204*	0.56811498	*0.56792204*
	(98 func evals)	(73 func evals)	(102 func evals)
UOBYQA	0.57113598	0.56792960	0.56792235
	(95 func evals)	(136 func evals)	(80 func evals)
Wedge	0.56793007	0.56792988	0.56814486
	(48 func evals)	(51 func evals)	(46 func evals)

Table 9: Search method results for the second real data problem.

$25 + 122 = 147$; the maximum total function evaluations on the 10×10 grid was $100 + 136 = 236$. Based on these results, we suggest using a coarse 5×5 grid search to obtain a good starting point, then applying one of the derivative-free methods to finish the outer optimization. Any of the three methods could be used. However, the Nelder-Mead method found the best point in the two real data problems. The Wedge method though, required fewer function evaluations (generally around half).

As the tables show, improved starting points do not always result in better solutions. For example, the UOBYQA method is worse at the 10×10 grid starting point than at the 5×5 grid starting point on the first real problem. Similarly, the Nelder-Mead method is worse at the 10×10 grid starting point than at the 5×5 grid starting point on the second real data problem. Also on this problem, the Wedge method actually obtains the worst solution of the derivative-free methods at the 20×20 grid starting point. These results suggest that a starting point farther from the solution may result in a better model being built and thus a better final value. (The trust region methods may also be improved by larger initial trust region radii.)

Chapter 4

Treatment Planning

In the previous two chapters, we have looked at improving the efficiency of large scale mathematical programming problems. Small versions of the problems presented thus far are easily solved using standard methods. In this chapter, we consider a problem which is difficult to solve even in small cases. As a result, we turn from solving exactly to approximating solutions.

In radiation therapy, ionizing radiation is applied to cancerous tissue, damaging the DNA and interfering with the ability of the cancerous cells to grow and divide [91, 108]. Healthy cells are also damaged by the radiation but they are more able to repair the damage and return to normal function. Since both cancerous and healthy cells are affected by radiation, dose distributions need to be designed that expose the tumor to enough radiation for treatment while, at the same time, avoid excessive radiation to surrounding healthy tissue and in particular, nearby organs.

Given a particular delivery mechanism, a treatment plan corresponds to settings of the machine that facilitate the delivery of the target dose distribution. Optimization techniques can be used to design such plans [14,

16, 84, 94]. Typically these problems are complicated due to the ever increasing complexities of the delivery mechanisms [13] and the large amount of data that needs to be manipulated to obtain sufficient detail of the dose on the target area. While these problems remain at the forefront of cancer treatment planning and many techniques have been proposed for the large varieties of machines (for example, see [26, 35, 36, 44, 47, 59, 73]), many of which take from minutes to hours to solve, we will not focus on this aspect of the problem. Instead, as we now describe, we will look at the day-to-day planning problem and derive target distributions that hedge against errors in the delivery process and assume the aforementioned planning tools will be used on specific machines to approximate these target distributions.

The day-to-day planning problem arises since many cancer patients are treated by a course of radiation over a period of days or weeks. For example, the full dose may be delivered in 20 or so treatments, with 1/20-th of the total dose delivered at each stage. This limits burning and gives the healthy tissue time to recover. As mentioned above, particular planning tools approximate the idealized dose, leading to errors between the planned and delivered dosage. Furthermore, in dividing the radiation dose over a series of treatments, additional errors can be introduced. Many sources can contribute errors to individual treatments, including the re-registration of the patient on the machine, the movement of the patient during treatment, and machine error [39, 103]. Such errors can result in dose displacement, leading

to loss of tumor control and injury to normal or sensitive tissue nearby [71]. Thus, it is important that we account for such errors.

At this time, mechanisms to determine the actual dose delivered during individual treatments are quite primitive. More advanced imaging devices are currently being developed that can generate more accurate delivery information, highlighting where the delivered treatment may be inaccurate. The purpose of this chapter is to exploit this knowledge to improve the overall treatment.

The chapter aims to generate a deliverable plan for each treatment in the course that compensates over time for movement of the patient and error in the delivery process. We develop a control mechanism for the treatment course, leaving the implementation of the daily dosage to a specialized planning tool. To find the control, we use neuro-dynamic programming, particularly a rollout policy, to improve upon simple heuristic policies.

In the next section, we describe approaches from the literature for dealing with uncertainty in radiation treatment. Then we consider the mathematical framework in which we will be working and describe techniques for solving the day-to-day planning problem. These techniques include neuro-dynamic programming (NDP) and heuristic policies, one of which is the current method of choice. We next present examples and discuss their results, showing how the NDP ideas can improve upon the heuristic policies. We define rules of thumb, which allow for practical implementations of solutions suggested by

NDP while still maintaining most of the improvements. Finally, we show how a simple rule of thumb obtained from the NDP approach performs on actual patient data under replanning.

4.1 Approaches to Uncertainty in Radiation Treatment

The traditional method to deal with uncertainty and patient movement in radiation treatment is to place a margin around the tumor and consider the resulting volume as the target [49, 51, 50]. Under this approach, we begin with a clinical target volume (CTV) determined by adding a margin to the gross tumor volume (GTV) to take into consideration "potential 'subclinical' invasion" [50]. The CTV is an oncological concept that specifies the pure target. From the CTV, a planning target volume (PTV) is established. To obtain the PTV, an internal margin (IM) and a set-up margin (SM) are combined in some form with the CTV. The IM is defined to take into consideration physiologic variations; the SM is defined to take into consideration uncertainties in technical factors such as patient set-up and mechanical stability [50]. As noted in [51, 50], simply adding the CTV, IM and SM to obtain the PTV may result in an excessively large PTV that could result in normal tissue complications. Thus, typically a global safety margin is defined that depends upon the situation at hand: when sensitive structures (organs

at risk) are nearby, a smaller margin is used. The definition of the safety margin usually involves a compromise. For example for organs at risk, a planning organ at risk volume (PRV), analogous to the PTV for the target, is defined [51, 50]. When the PRV and the PTV overlap, then the safety margin becomes a compromise between the two volumes, as determined by the radiation-oncology team [50]. The advantage to this approach is that small displacements can occur and the tumor will still be dosed. However, normal tissue and/or organs at risk will also be dosed, due to the use of the margins.

Besides considering PTV margins, other clinical methods have been investigated [111]. Kubo and Hill [58] look at synchronizing the radiotherapy beam with respiration to minimize patient movement during treatment. Wong et al. [110] study active breathing control, involving immobilization of the patient to minimize movement during treatment. Keall et al. [56] consider motion adaptive x-ray therapy, which adapts the beam to follow target motion. All of these clinical techniques focus on restricting internal movement due to respiration, but do not consider set-up issues like patient registration errors.

Statistical approaches have also been presented to deal with uncertainty in radiation treatment. Löf, Lind and Brahme [71] address the issue of modifying radiation beams to minimize the difference between the desired dose

and the delivered dose, under the assumption that uncertainty in the process is governed by a known stochastic process. They build on the work of Lind et al. [68] (who earlier had solved the problem for special cases) by using symmetry and numerical integration techniques for small problems, and a Monte Carlo integration method for general cases. Similarly, Li and Xing [64] consider reducing the "hard margins" of PTV by representing random organ motion in terms of a spatial probability distribution, specifically a three-dimensional Gaussian. For fractionated stereotactic radiotherapy (radiotherapy with relocatable fixed beam heads), Zavgorodni [114] modifies the margin approach by convolving the dose with the probability density distribution of the the isocenter (where the beams meet).

Other statistical approaches use feedback from the system to adjust beam profiles. Löf, Lind and Brahme [72] incorporate dynamic optimization techniques into their work of [71]. Internal variations (such as organ movement) are dealt with using stochastic optimization, as before. Dynamic optimization though, is added to automatically adjust beam profiles and patient location in subsequent treatments to account for current set-up errors. Re-optimization has also been explored. In [113], Yan et al. discuss the conceptual idea of re-optimizing by adapting the margins treatment. Wu et al. [111] generalize this by also modifying the original treatment plan.

The procedure we present also requires feedback from the system. But rather than modifying a predetermined treatment plan, we build the plan

as treatments progress, focusing on the total dose to be delivered. Further, our procedure does not depend upon a particular delivery mechanism. Our focus is on determining appropriate doses to use; we leave the particular implementation to specific tools used by application experts for particular machines.

4.2 Model Formulations

To describe the problem more precisely, we introduce some notation and a simplified model that captures the salient features of the process. Let $\mathcal{I}$ be a collection of voxels (pixels, points) and let $T(i)$, $i \in \mathcal{I}$ represent the required final dosage (target). Suppose the course lasts N periods (stages), and the actual dose delivered (the state) after k days is x_k. This state evolves as a stationary discrete-time dynamic system:

$$x_{k+1} = f(x_k, u_k, w_k), \quad k = 0, 1, \ldots .$$

Here u_k is the control to be selected from a collection $U(x_k)$, and w_k is a random disturbance drawn from a set W. In the application, we assume that these random disturbances come from errors in the delivery process (such as patient movement) or errors in the setup (such as patient registration errors). For this reason, we assume that w_k corresponds to a shift to u_k. Further, since each treatment is delivered separately, the errors that arise pertain only to a particular treatment and time stage, and so w_k is independent over

stages. A key issue to note is that the controls are nonnegative since dose cannot be removed from the patient.

At the end of N stages, the state x_N should minimize a terminal cost G. For ease of exposition we assume that $G(x)$ is a linear combination of the differences between the current dose and the target at each voxel, that is

$$G(x) = \sum_{i \in \mathcal{I}} c(i) \left| x_N(i) - T(i) \right|.$$

Here, the vector c weights the importance of hitting the target value for each voxel. We typically use similar values of c for distinct areas in the target, such as the location(s) of the tumor, sensitive structures like organs, and normal tissue. In practice, larger values of c correspond to tumor areas and/or sensitive structures. This gives us the following mathematical model:

$$\begin{aligned} \min_{u} \quad & E(G(x_N)) \\ \text{subject to} \quad & x_{k+1}(i) = x_k(i) + u_k(i + w_k), \quad \forall i \in \mathcal{I}, k = 0, 1, \ldots, N-1 \\ & u_k \in U(x_k), u_k \geq 0, w_k \in W, \end{aligned} \tag{4.1}$$

with x_0 given (typically 0).

4.2.1 Stochastic Linear Programming

If W is a finite set, then the problem at hand can be formulated as a stochastic linear program whenever the constraints $u_k \in U(x_k)$ are linear relationships.

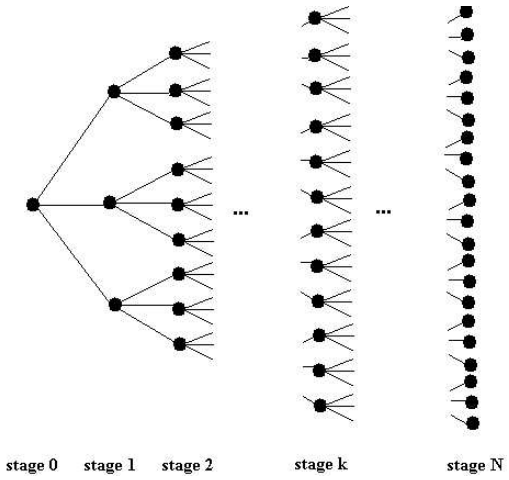


Figure 12: Scenario tree S for the application.

To explain this, assume that at each time stage k, one of $|W|$ scenarios occurs. If we align the time stages on a horizontal axis, the resulting scenario tree S can be depicted as in Figure 12, where as an example we have taken $|W| = 3$. Let $V(k)$ represent the nodes n in this tree S belonging to time stage k, while $p(n)$ denotes the predecessor node of n, i.e., if $n \in V(k)$, then $p(n)$ is the unique element of $V(k-1)$ such that $(p(n), n) \in S$. For this section, we introduce a slight abuse of notation. We use $x(n, \cdot)$ and $u(n, \cdot)$ to denote the state and control at a node n, whereas $x_k(\cdot)$ and $u_k(\cdot)$ denote the state and control at time stage k.

Using this notation, at each node n in the scenario tree,

$$x(n, i) = x(p(n), i) + u(p(n), i + w(p(n), n))$$

where $w(p(n), n)$ is the shift that is applied to u as we move from $p(n)$ to n.

Then, model (4.1) can be reformulated as the following stochastic program:

$$\begin{aligned} \min \quad & \sum_{n \in V(N)} \Pr(n) \sum_{i \in \mathcal{I}} c(i) \, |x(n,i) - T(i)| \\ \text{subject to} \quad & x(n,i) = x(p(n),i) + u(p(n), i + w(p(n),n)), \\ & \qquad \forall n \in V(k), k = 1, \ldots, N, i \in \mathcal{I} \\ & u(n,i) \geq 0, \quad \forall n \in S, i \in \mathcal{I} \end{aligned} \tag{4.2}$$

with $\Pr(n)$ depicting the probability of being at node n. Here we have taken just simple nonnegativity on u; more complex linear relationships are also feasible. Standard techniques can be used to reformulate (4.2) as a linear program. When N is small and the number of possible shifts are small, then a wealth of techniques for such problems can be applied [12, 54]. However, when $\mathcal{I}$ or N becomes large, the size of the problem soon becomes too great, and we have to resort to approximation schemes for the solution.

For the prototype examples of Section 4.3 where we have $|I| = 9$ and $N = 4, 5, 6, 10, 14$ and 20, we attempted to solve model (4.2) exactly. We formulated the model in the GAMS [17] modeling language and used the CPLEX [48] barrier method to obtain the solutions (since this significantly outperformed all simplex options). Even with small numbers of time stages, the problems became quite large. With 4 time stages, the model consisted of 18271 equations, 14059 variables and 48250 nonzeros; solution times were

around 25 seconds. With 5 time stages, the model consisted of 91396 equations, 70309 variables and 241375 nonzeros; solution times ranged from about two-and-a-half minutes to a little over 3 minutes. With 6 time stages, the model consisted of 457021 equations, 351559 variables and 1207000 nonzeros; solution times ranged from about one-half hour to around 70 minutes. Due to the storage requirements, this became intractable for more than 6 time stages. While more powerful machines would extend the number of stages somewhat, the exponential growth precludes solutions for $N = 10, 14$ or 20. It may be possible to apply scenario reduction or sampling techniques to this model [46, 69], but this was not explored here.

4.2.2 Dynamic Programming

Dynamic programming is another method that can be used to solve model (4.1). Since each decision impacts later decisions, the choice of control at each time stage contributes to the final cost. Thus, to find the optimal control at each stage, we must consider future states. Under dynamic programming, we apply backward recursion: start at the last stage and determine the optimal control to apply for each state; then consider the second-to-last stage, and so on, working backward through the stages.

As a means of determining the optimal control for a state, we consider the costs-to-go of various policies from the current state. The cost-to-go for a particular policy is the optimal cost over all remaining stages, starting from

the current state and applying the given policy. Starting in stage k from state x_k and using the policy $\pi = \{u_0, u_1, \ldots, u_{N-1}\}$, the cost-to-go is [10]:

$$J_k(x_k) = E[G(x_N) + \sum_{i=k}^{N-1} g(x_i, u_i(x_i), w_i)],$$

where $g(x_i, u_i(x_i), w_i)$ represents the immediate cost of applying control $u_i(x_i)$. As noted in [10, 11], the cost-to-go functions satisfy the dynamic programming recursion

$$J_k(x) = E[g(x, u_k(x), w) + J_{k+1}(f(x, u_k(x), w))] \tag{4.3}$$

with initial condition

$$J_N(x) = G(x). \tag{4.4}$$

However, since we are attempting to construct the optimal policy $\bar{\pi}$, the individual controls $\bar{u}_k$ are not known a priori. These controls can be found by backward recursion by considering the costs-to-go over all possible controls:

$$\bar{u}_k(x) = \arg \min_{u \in U(x)} E[g(x, u, w) + J_{k+1}(f(x, u, w))]. \tag{4.5}$$

If we wanted to use equations (4.3), (4.4), and (4.5) in the radiation treatment application, we would need to calculate $J_k(x)$ and $\bar{u}_k(x)$ for every possible state x at each stage k. A standard technique for doing this is to discretize the state space for x and form a lookup table for J_k and u_k over this discretization. For each voxel in the target, we would need a minimum of N discretizations to allow for a simple heuristic policy, the constant policy

(described later in Section 4.2.4), to be implemented in the lookup table. Even for the simple examples that we describe in Section 4.3, this becomes unmanageable.

4.2.3 Neuro-Dynamic Programming (NDP)

Another technique to approximately solve (4.1) is to apply a rollout policy [10, 11]. Rather than starting at stage N, we start at stage 0 and work our way forward, determining the policy $\bar{\pi}$ one stage at a time (we "roll" out the policy). This is much closer to the way a decision maker would work in practice: only today's decision is needed precisely; the remaining decisions can be approximated. The rollout policy uses equation (4.5) to find the control $\bar{u}_k$ to apply at each stage k, but using an approximation to J_{k+1} instead of the actual function. This approximation is built by applying the particular control u at stage k and a control from some base (heuristic) policy at all future stages. Then, J_{k+1} is calculated using these controls and methods such as simulation (which we use) or neural networks.

In effect, this is an example of an on-line policy choice. In the practical setting, we approximate the future by simulation and choose the policy to apply right now by optimization. After applying this policy, we wait for time to elapse and repeat the same process at the next stage. For a particular radiation therapy, the choice of the current control may itself involve a lengthy optimization, and the error produced in delivery will only be provided to

the decision maker after the current control is delivered (but before the next control is calculated).

In the radiation treatment application that is expressed in model (4.1), we assume no immediate costs in applying individual controls and so $J_k(x_k) = E[G(x_N)]$. To apply the rollout policy, we begin by choosing the base policy for the calculation of the costs-to-go J_{k+1}. As the base policy is applied at all later stages, it should be a heuristic policy that performs well (we use the reactive policy, described in Section 4.2.4). Then we simulate the effects on the system as time advances. In principle, we calculate for each one-stage control $u_k \in U(x_k)$ the Q-factor,

$$Q_k(x_k, u_k) := E[g(x, u, w) + J_{k+1}(f(x, u, w))]$$

using simulation by applying u_k at stage k and the reactive policy at all later stages. The (approximated) Q-factor for each control is then the expected terminal cost from the simulation using that control at the current time stage.

To choose between controls, we need to evaluate differences between $Q_k(x_k, u)$ for each $u \in U(x_k)$. Since simulation is involved, this will be prone to errors. These errors can be alleviated somewhat using the same realization of w when calculating all Q-factor differences [10]. Thus, we calculate the average differences in the expectations for every $u \in U(x_k)$. Note that all the controls can be compared directly like this as individual controls are only applied at the current time stage. By applying the base policy at all future time stages, we can test the effectiveness of each control against

the others for the current stage. In addition, since the controls are applied simultaneously, they are applied under the same shifts and so the comparison is done for the same realization of w.

4.2.4 Heuristic Policies

One approach to overcome the computational burden outlined above is to apply heuristic policies. Besides offering an alternative to the techniques described above, we also use heuristic policies to define the (finite) set $U(x_k)$ for the NDP rollout approach.

Several heuristic control techniques immediately spring to mind. First, there is the simple plan to deliver

$$u_k := T/N$$

at each stage and not account at all for disturbances in the delivery. This plan can be used when treatment errors cannot be measured directly, and is currently the method of choice. We refer to this plan as the constant policy. Note that the implementation on a particular machine of this policy only needs one optimization to be performed at the start of the process. However, even when a voxel has been overdosed at stage k, the constant policy continues to add dose at subsequent time stages. An alternative is to only add dose if the current dose is less than the target dose. We refer to this modification as the constant-plus policy. Surprisingly, the simulations

show that this has little effect on overall error.

An alternative to the constant policies is to attempt to compensate for the error delivered in the previous time by spreading the error over the remaining time stages. At each time stage, we divide the residual over the remaining time stages:

$$u_k := \max(0, T - x_k)/(N - k).$$

We refer to this plan as the reactive policy. Since the reactive policy takes into consideration the residual at each time stage, we expect that the reactive policy will perform better than the constant policies. Note, though, that the reactive policy requires knowledge of x_k and replanning at every stage k.

We show later in this paper how the constant and reactive heuristic policies perform on a variety of examples. We also show how the NDP rollout approach improves upon these results.

To apply the NDP rollout approach, we require a rich collection of heuristics for the finite set $U(x_k)$. We use the constant, constant-plus, and reactive policies, but we also use what we refer to as categorical policies. For these policies at stage k, we calculate the residual target for each voxel i by $\max\{0, T(i) - x_k(i)\}$. Then, the voxels are divided into three categories by comparing their residual target to the maximum residual:

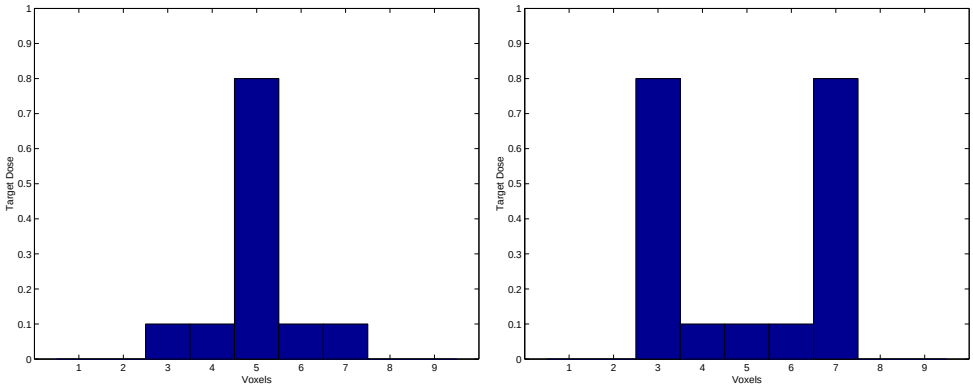
$$\max_{i \in \mathcal{I}} \max\{0, T(i) - x_k(i)\}.$$

The three categories correspond to voxels whose residual target is less than

40% (low residual), between 40% and 70% (medium residual), and greater than 70% (high residual) of this maximum value. In each category, we apply one of three controls. Either we apply 0 dosage, 0.4 of the residual target, or $1/(N-k)$ of the residual target. This yields an additional 26 policies (as the reactive policy is the categorical policy with $1/(N-k)$ applied in each category).

Note that the practical implementation of a policy generated by NDP using these controls for $U(x_k)$ is exactly the same as that of the reactive policy. First of all, knowledge of x_k is required. Given this information we can calculate the actual dose that should be delivered at each voxel $i \in \mathcal{I}$ by determining which category the voxel resides in, and then multiplying the residual $T(i) - x_k(i)$ by the categorical multiplier. Knowing the dose at every voxel i is all the data that is required to specify a plan optimization that determines how to implement that particular dose on a specific machine. As mentioned in the introduction, we allow existing planning tools to perform this step, and we believe this is a key advantage of our approach.

In actual treatment plans, individual doses are subject to an upper bound, applied in order to limit burning and allow for healthy tissue to recover between treatments. For this reason, we assign a cutoff value that restricts the dose prescribed by each control to such an upper bound. For testing purposes, we set the cutoff to be $2T_{\max}/N$, which is double the dose prescribed by the constant policy in the worst case. Such a value allows for a large dose



(a) Spike target (b) Double spike target

Figure 13: Example targets

to be prescribed, while still ensuring that the dose is not unreasonably large. Although this is chosen for the application, very little changes if this upper bound is not applied.

4.3 Examples and Results

We consider two simple, one-dimensional targets under different weighting and probability distributions, pictured in Figure 13.

For both targets, $\mathcal{I} = \{1, 2, \ldots, 9\}$ and we allow a maximum shift of 2 voxels. In both targets, the "spikes" of dose 0.8 represent tumor locations. Thus, it is important that these areas receive as much of the 0.8-prescribed dose as possible, and so these areas will have a relatively high weighting in the objective. The 0.1 areas can represent sensitive structures (which should

be exposed to a minimum amount of radiation) or normal tissue, depending upon the particular weighting scheme employed. We apply 3 different weighting schemes to the spike target. Moving from easiest to hardest, these schemes are:

- the smooth weighting:

$$c = [1, 1, 1, 1, 10, 1, 1, 1, 1],$$

  which only enforces the 0.8-dosage, allowing for more variation in the other voxels (including a "building" up to the spike);

- the nonsymmetric weighting:

$$c = [1, 1, 1, 1, 10, 5, 5, 1, 1],$$

  which allows for a build-up to the spike on the left-hand-side, but enforces the spike structure more rigidly on the right-hand-side; and

- the spike weighting:

$$c = [1, 1, 5, 5, 10, 5, 5, 1, 1],$$

  which enforces the spike structure rather rigidly.

For the double spike target, we apply the double spike weighting scheme:

$$c = [1, 1, 10, 5, 5, 5, 10, 1, 1],$$

which enforces high dosage on the target edges and the low dosage in the center. The examples have been chosen to simulate practical cases of interest in the application area.

For the targets above, we also consider three different probability distributions for the shifts. The low volatility examples have

$$w_k = \begin{cases} -2 & \text{with probability } 0.02 \\ -1 & \text{with probability } 0.08 \\ 0 & \text{with probability } 0.8 \\ 1 & \text{with probability } 0.08 \\ 2 & \text{with probability } 0.02. \end{cases}$$

for every stage k. The medium volatility examples have

$$w_k = \begin{cases} -2 & \text{with probability } 0.05 \\ -1 & \text{with probability } 0.15 \\ 0 & \text{with probability } 0.6 \\ 1 & \text{with probability } 0.15 \\ 2 & \text{with probability } 0.05. \end{cases}$$

for every stage k. The high volatility examples have

$$w_k = \begin{cases} -2 & \text{with probability } 0.05 \\ -1 & \text{with probability } 0.25 \\ 0 & \text{with probability } 0.4 \\ 1 & \text{with probability } 0.25 \\ 2 & \text{with probability } 0.05. \end{cases}$$

for every stage k. While it is hard to estimate the volatilities present in the given application, our results of Section 4.4 are fairly insensitive to these choices.

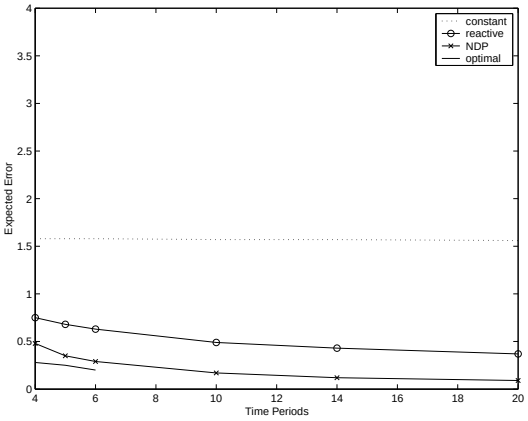
As described in Section 4.2.3, we use a simulation code at every stage to determine the optimal $\bar{u}_k \in U(x_k)$ by calculating differences in Q-factors under the same realizations w. The simulation code we use generates 10000 paths through the simulation tree between stage k and stage N. For each path, the Q-factor differences are calculated; at the end, the average of these differences determines $\bar{u}_k$. The same code can be used to simulate the costs of the individual heuristic policies. Essentially, for this we ensure that $U(x_k)$ is the appropriate singleton.

While we described how to develop the rollout policy in an on-line fashion in the previous section, we also need to evaluate the effectiveness of our procedure. To effect this, we apply an outer simulation that simulates paths

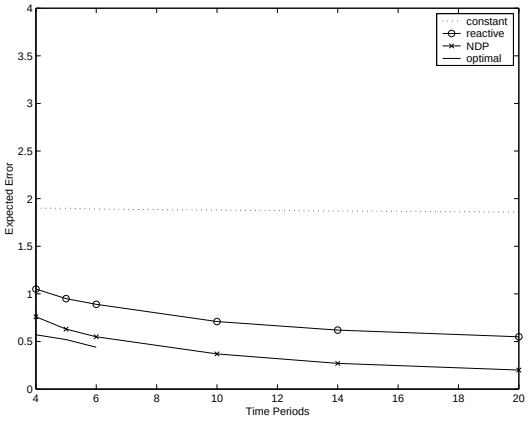
through the scenario tree. The outer simulation evolves one stage at a time, therefore assuming that x_k is known at each stage k. We use the inner simulation (for Q-factor differences) to determine $\bar{u}_k$. The outer simulation then generates w_k and thus forms x_{k+1}. After N stages, x_N is known and the terminal cost can be evaluated for this particular path through the scenario tree. The outer simulation generates 20000 paths to form an expected value for the terminal cost.

Running the outer simulation (with repeated inner simulations needed inside), results in a great deal of computation and long running times. To deal with this efficiently, we submitted the outer simulations to Condor [70], a network resource manager. Once a job is submitted to the Condor queue, Condor searches for idle network machines. If one is found, then the simulation starts executing on that machine; otherwise the simulation is held until sufficient resources are freed. In addition, Condor migrates jobs or checkpoints them (for later continuation) when the machine's owner returns or resources become scarce.

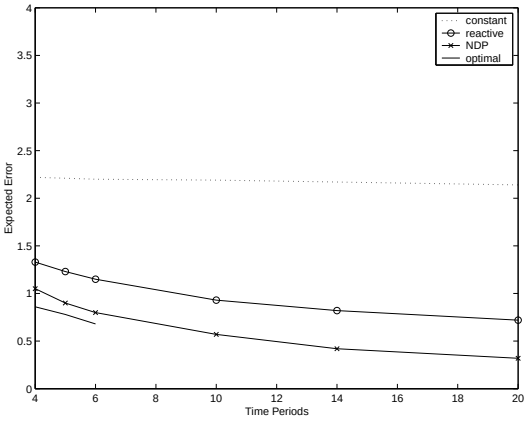
Figures 14, 15, and 16 display simulated results for each example under the three probability distributions. For each graph, the constant policy, reactive policy, and NDP rollout policy results are displayed, as well as the optimal results for time stages 4, 5, and 6. The optimal results come from solving model (4.2) exactly, as explained in Section 4.2.1. Note the change of vertical scales between the three figures.



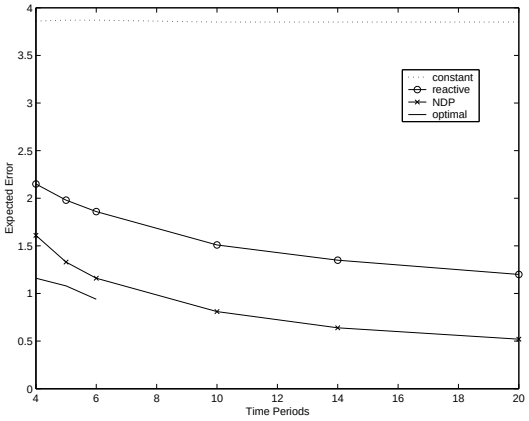
(a) Spike target with smooth weighting.



(b) Spike target with nonsymmetric weighting.

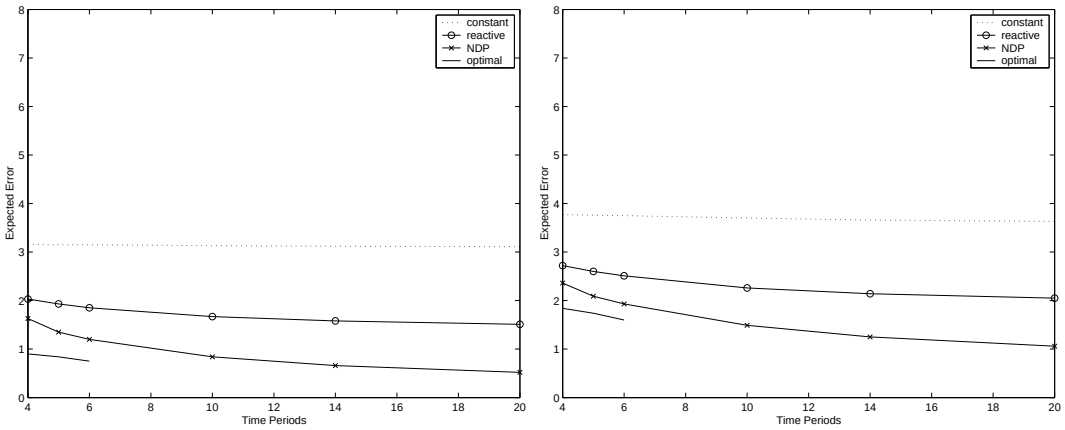


(c) Spike target with spike weighting.



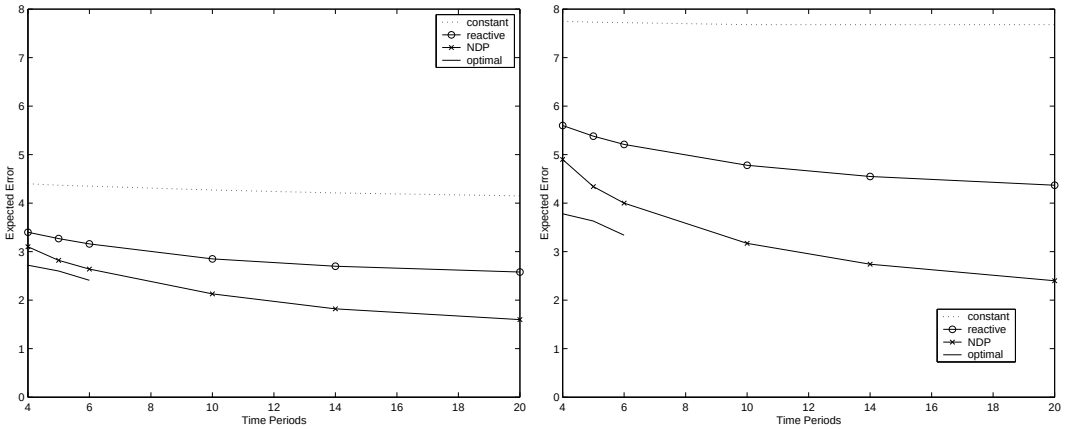
(d) Double spike target with double spike weighting.

Figure 14: Examples under low volatility.



(a) Spike target with smooth weighting.

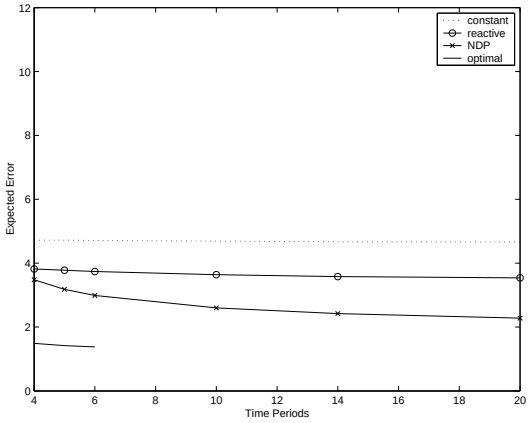
(b) Spike target with nonsymmetric weighting.



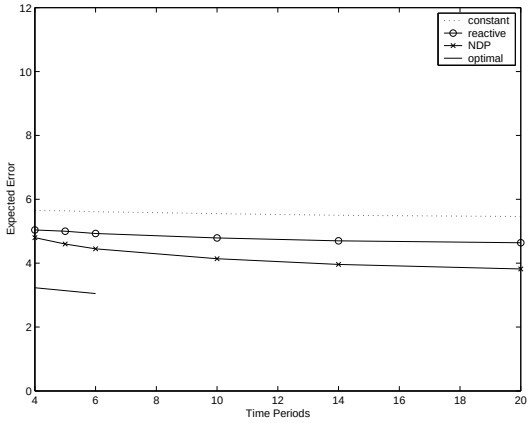
(c) Spike target with spike weighting.

(d) Double spike target with double spike weighting.

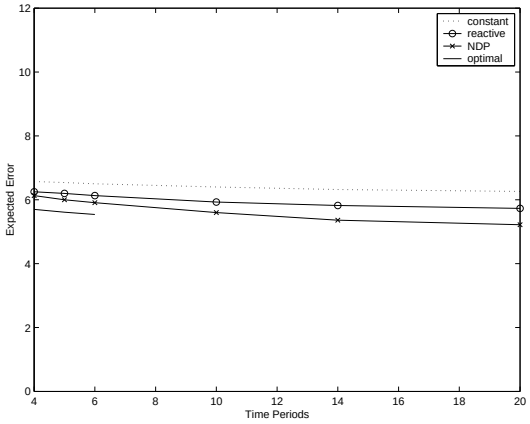
Figure 15: Examples under medium volatility.



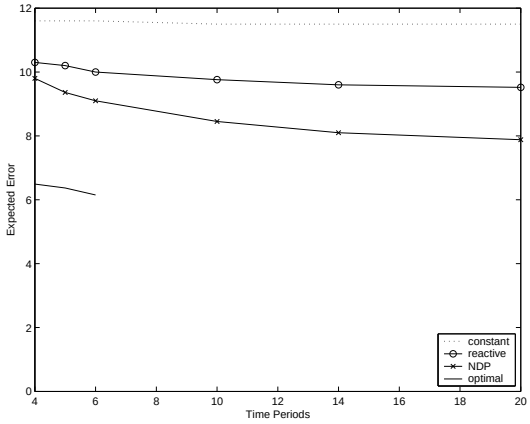
(a) Spike target with smooth weighting.



(b) Spike target with nonsymmetric weighting.



(c) Spike target with spike weighting.



(d) Double spike target with double spike weighting.

Figure 16: Examples under high volatility.

Comparing Figures 14, 15, and 16, we note the remarkable similarities. While the vertical scales are larger as the volatility increases, general conclusions are easy to draw. Firstly, the alphabetic ordering of targets (a) to (d) are increasingly difficult and lead to larger errors, independent of the optimization scheme chosen. Secondly, in all cases and for all optimization schemes, as volatility increases, so does the error.

Common to all examples is the poor performance of the constant policy. The reactive policy performs better than the constant policy, but not as well as the NDP rollout policy. The level of improvement though, depends upon the difficulty of the target and the volatility. In the low and medium volatility spike examples, the NDP results are much closer to the optimal results than in the high volatility examples, particularly in the double spike example (Figure 16(d)). However, the NDP results decrease at a faster rate than the optimal results. Thus, more time periods are beneficial. These decreases level off at later time stages, exhibiting decreasing returns for more time stages.

Focusing on the low volatility examples of Figure 14, we see that the reactive policy gives a large improvement over the constant policy — the error is nearly halved. NDP does even better, yielding about a 50% drop in the reactive policy error at larger time stages (the exact improvements over the constant policy are given in Table 16), and achieving near-optimal results at smaller time stages. As time advances, the improvement for both the NDP

and reactive policies becomes greater: where constant remains almost level, reactive and NDP continue to drop as we move to later time stages. Further, NDP decreases faster than the optimal results do, suggesting that it may become optimal at later time stages.

The medium volatility examples of Figure 15 show less improvement in the NDP results. Although the constant policy appears level, the reactive policy gives slightly less improvement — not quite 50%. We also see slightly less than a 50% drop in the NDP results over the reactive policy results at later time stages, due to its faster rate of decrease. Although the NDP results are not as close to the optimal results as in the low volatility examples, again we see that the NDP error decreases faster than the optimal error, suggesting that the NDP policy may be close to the optimal error at some later time stage.

The high volatility examples of Figure 16 show the greatest errors of all of the examples. Again, the constant policy appears to be level and is much larger than in the previous examples. We see less improvement in the NDP results: in Figures 16(b) and 16(d), the NDP improvement is approximately one-third over the constant policy; Figure 16(a) is better (about one-half improvement), but Figure 16(c) is worse (the constant results, though, are much closer to optimal here). Apart from this latter case, the NDP results are far from the optimal results, lying closer to the reactive policy than to the optimal policy. We see that the NDP results do improve faster than the

reactive results so this may change at later time stages.

The high volatility examples are the most difficult of the examples; in these examples, we are more likely to see an error shift than not. As a result, although we use information regarding earlier errors, it is difficult to account for future errors. The optimal results show that, given unlimited possibilities for policy choices, we can often improve greatly upon the currently-used constant policy. However, as NDP is limited by a finite number of policy choices — in particular, a choice between 0, $1/(N-k)$ or 0.4 of the current target residual — it is more difficult for the NDP policy to achieve such substantial improvements.

In addition to three-category policy choices, we also experimented with two-category policy choices. Under these policies, the voxels were classified as either less than 50% of the maximum residual or more than 50% of the maximum residual. To maintain approximately the same number of policies, we allowed five choices for each category (resulting in a total of 27 policies, including the constant policies). In one experiment, we allowed for small multiples: 0, $1/(N-k)$, 0.01, 0.1, or 0.4. In another experiment, we allowed for large multiples: 0, $1/(N-k)$, 0.1, 0.4, or 0.6. Applying these policies to the double spike example (the hardest example), we found very little change in the NDP results. The small-multiple category choices returned approximately the same results as the three-category choices, while the large-multiple category choices returned slightly better results but nothing visually

significant on the plot.

These results suggest that significant improvements over the presented NDP results cannot be achieved while choosing from among approximately 30 policies. Enriching the policy set by combining the two-category policies with the three-category policies, or moving to five-category policies (for example) seems to be the only way to improve upon the NDP results. Other policies may come from previous real-life plans or other planning systems.

Note that in addition, we also experimented with many more examples, including different targets, different weighting schemes, and larger targets. The results from these other examples were qualitatively the same. We did find though, that for high volatility examples, constant weighting ($c(i) = 1, \forall i \in \mathcal{I}$) resulted in significantly underdosing the target. This strongly suggests that the use of an appropriate weighting scheme to focus the treatment is imperative.

4.4 Off-Line Planning

While building simple models and analyzing their properties can lead to great insight into the application at hand, it is important to draw definitive conclusions that are applicable to the real problem. In this section, we endeavor to derive policies that are directly implementable in the radiation treatment planning arena.

Besides testing the NDP model, the real-life (outer) simulation is also useful for off-line planning. In Section 4.3, we compared the on-line planning schemes, that is, we assumed that the controls were determined in between treatments as we moved to the next stage. Only at the end of the treatment period would we have a complete policy. Off-line planning, on the other hand, assumes that a policy is pre-defined, prior to the beginning of treatment.

To find pre-defined policies, we look for policies that are good for most, if not all, of the examples. For a particular example, the outer simulation gives a series of possible policies to apply. By considering the average Q-factors for each control, we have an idea of how effective that control is for that example at that time stage. Averaging these Q-factors across examples with the same volatility and choosing the controls that correspond to the smallest Q-factors at each time stage, we determine a generalized policy for each volatility. We refer to this generalized policy as the "rule of thumb" policy for that volatility. These rules of thumb allow us to remove the target dependence from the simulation and also provide us with a pre-defined plan to use for a particular volatility.

We can take the generalization further and remove dependence on the volatility by averaging the Q-factors across volatilities as well. We refer to the resulting policy as the "simple rule of thumb". Since the total number of time stages N affects which controls are chosen and when, we define rules of thumb and simple rules of thumb for each N. The rules of thumb and

Stage	Low Volatility	Med. Volatility	High Volatility	Simple Rule
1–2	$(0, 0, 0.4)$	$(0, 0, 0.4)$	$(0.4, 0.4, 0.4)$	$(0, 0, 0.4)$
3	(0,0.4,0.5)	$(0.5, 0.5, 0.5)$	$(0.5, 0.5, 0.5)$	$(0, 0.4, 0.5)$
4	$(1, 1, 1)$	$(1, 1, 1)$	$(1, 1, 1)$	$(1, 1, 1)$

Table 10: Rules of thumb for 4 time period examples.

Stage	Low Volatility	Med. Volatility	High Volatility	Simple Rule
1–3	$(0, 0, 0.4)$	$(0, 0, 0.4)$	$(0.4, 0.4, 0.4)$	$(0, 0, 0.4)$
4	(0,0.4,0.5)	$(0.5, 0.4, 0.5)$	$(0.5, 0.5, 0.5)$	$(0.5, 0.4, 0.5)$
5	$(1, 1, 1)$	$(1, 1, 1)$	$(1, 1, 1)$	$(1, 1, 1)$

Table 11: Rules of thumb for 5 time period examples.

simple rules of thumb for $N = 4, 5, 6, 10, 14$ and 20 are given, respectively, in Tables 10, 11, 12, 13, 14 and 15. In these tables, the categorical policies (including the reactive policy) are given as triplets. In these triplets, the first entry corresponds to the low residual areas; the second entry corresponds to medium residual areas; and the third entry corresponds to the high residual areas. These entries correspond to the multiplier of the residual that is used at all voxels in that area.

Note that if the policy pool U is changed, the simulations must be rerun and this process must be repeated on the new results in order to determine appropriate rules of thumb and simple rules of thumb.

Stage	Low Volatility	Med. Volatility	High Volatility	Simple Rule
1	$(0, 0, 0.4)$	$(0, 0, 0.4)$	$(0.4, 0.4, 0.4)$	$(0, 0, 0.4)$
2–3	$(0, 0, 0.4)$	$(0, 0, 0.4)$	$(0, 0, 0.4)$	$(0, 0, 0.4)$
4	$(0, 0, 0.4)$	$(0, 0, 0.4)$	$(0.4, 0, 0.4)$	$(0, 0, 0.4)$
5	$(0.5, 0, 0.5)$	$(0.5, 0, 0.5)$	$(0.5, 0.5, 0.5)$	$(0.5, 0, 0.5)$
6	$(1, 1, 1)$	$(1, 1, 1)$	$(1, 1, 1)$	$(1, 1, 1)$

Table 12: Rules of thumb for 6 time period examples.

Stage	Low Volatility	Med. Volatility	High Volatility	Simple Rule
1–5	$(0, 0, 0.4)$	$(0, 0, 0.4)$	$(0, 0, 0.4)$	$(0, 0, 0.4)$
6	$(0, 0, 0.4)$	$(0, 0.2, 0.4)$	$(0, 0.2, 0.4)$	$(0, 0.2, 0.4)$
7	$(0, 0.25, 0.4)$	$(0, 0.25, 0.4)$	$(0, 0.25, 0.4)$	$(0, 0.25, 0.4)$
8	$(0, 0.33, 0.4)$	$(0.33, 0.33, 0.4)$	$(0.4, 0.33, 0.4)$	$(0.33, 0.33, 0.4)$
9	$(0.4, 0.4, 0.5)$	$(0.5, 0.5, 0.5)$	$(0.5, 0.5, 0.5)$	$(0.5, 0.5, 0.5)$
10	$(1, 1, 1)$	$(1, 1, 1)$	$(1, 1, 1)$	$(1, 1, 1)$

Table 13: Rules of thumb for 10 time period examples.

Stage	Low Volatility	Med. Volatility	High Volatility	Simple Rule
1–7	$(0, 0, 0.4)$	$(0, 0, 0.4)$	$(0, 0, 0.4)$	$(0, 0, 0.4)$
8	$(0, 0, 0.4)$	$(0, 0.14, 0.4)$	$(0, 0.14, 0.4)$	$(0, 0.14, 0.4)$
9	$(0, 0.17, 0.4)$	$(0, 0.17, 0.4)$	$(0, 0.17, 0.4)$	$(0, 0.17, 0.4)$
10	$(0, 0.20, 0.4)$	$(0, 0.20, 0.4)$	$(0, 0.20, 0.4)$	$(0, 0.20, 0.4)$
11	$(0, 0.25, 0.4)$	$(0, 0.25, 0.4)$	$(0, 0.25, 0.4)$	$(0, 0.25, 0.4)$
12	$(0, 0.4, 0.4)$	$(0.4, 0.4, 0.4)$	$(0.4, 0.33, 0.4)$	$(0.4, 0.4, 0.4)$
13	$(0.5, 0.5, 0.5)$	$(0.5, 0.5, 0.5)$	$(0.5, 0.5, 0.5)$	$(0.5, 0.5, 0.5)$
14	$(1, 1, 1)$	$(1, 1, 1)$	$(1, 1, 1)$	$(1, 1, 1)$

Table 14: Rules of thumb for 14 time period examples.

Examining the tables, we notice some general trends in the control choices. First of all, within each table, the controls become more aggressive as we near the final time stages, generally moving from controls in which only the high residual areas are dosed, to controls in which all areas are dosed. Typically, we use the first half of the time periods to work aggressively on the high residual areas and ignore the other areas. Exceptions to this are the high volatility rules for small time stages (4,5,6); in these cases, we apply dose to all areas. We conjecture that this comes from the fact that we are likely to make an error and we have very little time to correct it.

Controls in the middle stages vary but tend to focus on both the medium and high residual areas first. Later stages focus on all three categories, ending in every case aggressively with the reactive policy (to attempt to apply all

Stage	Low Volatility	Med. Volatility	High Volatility	Simple Rule
1–8	$(0, 0, 0.4)$	$(0, 0, 0.4)$	$(0, 0, 0.4)$	$(0, 0, 0.4)$
9	$(0, 0.08, 0.4)$	$(0, 0, 0.4)$	$(0, 0, 0.4)$	$(0, 0.08, 0.4)$
10	$(0, 0.09, 0.4)$	$(0, 0.09, 0.4)$	$(0, 0.09, 0.4)$	$(0, 0.09, 0.4)$
11	$(0, 0.1, 0.4)$	$(0, 0.1, 0.4)$	$(0, 0.1, 0.4)$	$(0, 0.1, 0.4)$
12	$(0, 0.11, 0.4)$	$(0, 0.11, 0.4)$	$(0, 0.11, 0.4)$	$(0, 0.11, 0.4)$
13	$(0, 0.125, 0.4)$	$(0, 0.125, 0.4)$	$(0, 0.125, 0.4)$	$(0, 0.125, 0.4)$
14	$(0, 0.14, 0.4)$	$(0, 0.14, 0.4)$	$(0, 0.14, 0.4)$	$(0, 0.14, 0.4)$
15	$(0, 0.4, 0.4)$	$(0, 0.17, 0.4)$	$(0, 0.17, 0.4)$	$(0, 0.17, 0.4)$
16	$(0, 0.4, 0.4)$	$(0, 0.4, 0.4)$	$(0, 0.20, 0.4)$	$(0, 0.4, 0.4)$
17	$(0, 0.4, 0.4)$	$(0, 0.4, 0.4)$	$(0.25, 0.4, 0.4)$	$(0.25, 0.4, 0.4)$
18	$(0.33, 0.4, 0.4)$	$(0.4, 0.4, 0.4)$	$(0.4, 0.4, 0.4)$	$(0.4, 0.4, 0.4)$
19	constant-plus	$(0.5, 0.5, 0.5)$	$(0.5, 0.5, 0.5)$	$(0.5, 0.5, 0.5)$
20	$(1, 1, 1)$	$(1, 1, 1)$	$(1, 1, 1)$	$(1, 1, 1)$

Table 15: Rules of thumb for 20 time period examples.

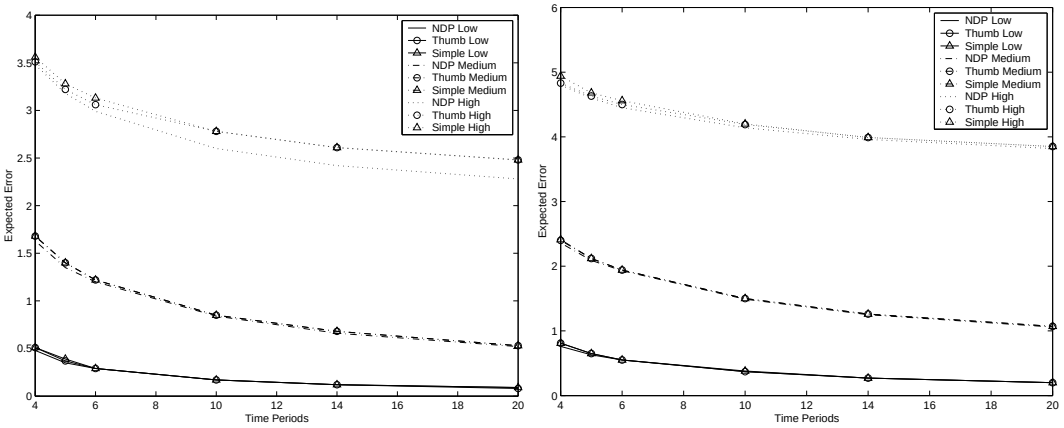
of the remaining dose). An interesting question arises as to whether the low volatility rule for $N = 20$ follows this general trend. In this case, the rule of thumb makes use of the constant-plus policy in the second-to-last time stage, where we would expect an aggressive policy to be employed. While $1/N$-th of the original target dose is a seemingly rather small amount, we claim that it is very likely to be an aggressive control here. This is because most of the earlier controls will have hit the target correctly (because of the low volatility), resulting in the remaining residual being small. Hence, the small fraction of the original dose is in fact a large dose in comparison to the required residual. Clearly, in this case, the removal of overdosing (the difference between constant and constant-plus) is important.

Focusing on the rules of thumb, the policies tend to use either the same or more aggressive controls for the same time stages as the volatility increases. Exceptions occur in the rules for $N = 20$ at stages stages 15 and 16, and in the rule for $N = 14$ at stage 12. In these cases, the low and medium volatility rules are more aggressive than the high volatility rules. Note that this is occurring near the end of the "middle" dosing period, where the low and medium volatility examples will probably have smaller error and thus less left to dose. The high volatility examples, on the other hand, will probably have made errors, and so are being more conservative prior to the aggressive push at the end. The simple rule policies tend to use either the same controls or combinations of the controls.

Target	Volatility	Reactive	NDP	RoT	SRoT
Smooth Spike	Low	76%	94%	95%	94%
Smooth Spike	Medium	51%	83%	83%	83%
Smooth Spike	High	24%	51%	47%	47%
Nonsymmetric Spike	Low	70%	89%	89%	89%
Nonsymmetric Spike	Medium	44%	71%	71%	71%
Nonsymmetric Spike	High	15%	30%	29%	29%
Spike Spike	Low	66%	85%	85%	85%
Spike Spike	Medium	38%	61%	61%	61%
Spike Spike	High	8%	17%	17%	17%
Double Spike	Low	69%	86%	86%	86%
Double Spike	Medium	43%	69%	68%	68%
Double Spike	High	17%	31%	31%	31%

Table 16: Percentage decrease over the constant policy at 20 time stages.

Since they are generalizations, we expect that the rules of thumb and the simple rule of thumb for each N will give perform worse than the (online) NDP rollout policy. However, we find that the differences tend not to be noticeable. Figure 17 compares the NDP rollout results to the rules of thumb and simple rules of thumb for each example. In addition Table 16 shows the percentage decrease achieved by the reactive policy, the NDP rollout policy, rules of thumb and simple rules of thumb over the currently-used constant policy at 20 time stages.



(a) Spike target with smooth weighting.

(b) Spike target with nonsymmetric weighting.

(c) Spike target with spike weighting.

(d) Double spike target with double spike weighting.

Figure 17: Rules of thumb and simple rules of thumb results.

The rules of thumb and simple rules of thumb are almost always indistinguishable from one another. Exceptions occur with small N, where we do not have enough time periods to make up for error from the generalization. Table 16 verifies the results shown in Figure 17. In all cases, the NDP, rules of thumb and simple rules of thumb improved upon the reactive policy results significantly. However, their percentage decrease over the constant policy varies very little between the three of them. This suggests that very little is sacrificed in moving to the generalized simple rules of thumb.

The results show that the simple rules of thumb, once determined, are almost as favorable as the NDP results. We find that the simple rule of thumb is effective for large complex target shapes (see Section 4.5), and we recommend its usage over both the constant and reactive policies. Certainly, the simple rule of thumb is no more costly than the reactive policy to implement and is shown to be much more effective at dealing with the errors that can arise in the planning process. If the resulting improvements are not sufficient for the treatment planning problem, then two further policies are suggested by the results of this paper. The first technique chooses a particular control structure and simulates to determine a simple rule of thumb, which can then be applied during the treatment process. A second more costly (but even more effective) approach is to generate the control policy using optimization within the on-line procedure of Section 4.2.3. In this setting, the treatment

planner is also able to choose a particular control structure. Under the second approach, the reoptimization need not be done at every time stage; for those time stages at which the reoptimization is not done, we can use the simple rule of thumb.

4.5 Real-Life Treatment Planning Example

In this section, we consider how well the NDP simple rule of thumb generalizes to a real-life example. The example that we use in this section is a three-dimensional example drawn from actual patient data.

As in the one-dimensional case, the area of consideration is divided into voxels. We consider an area where the x-axis varies from 0 to 110, the y-axis varies from 0 to 90, and the z-axis varies from 0 to 80. Within this area, there are $747,667$ voxels in normal tissue; $69,270$ voxels in four sensitive structures (spinal cord, liver, left and right kidneys); and $1,244$ voxels in the tumor. Figure 18 shows the layout at $z = 30$.

We consider a 20-stage treatment. Let $\mathcal{T}$ be the set of tumor voxels, $\mathcal{S}$ be the set of sensitive structure voxels, and $\mathcal{N}$ be the set of normal tissue voxels. Upon completion, we would ideally like to have the tumor dosed completely,

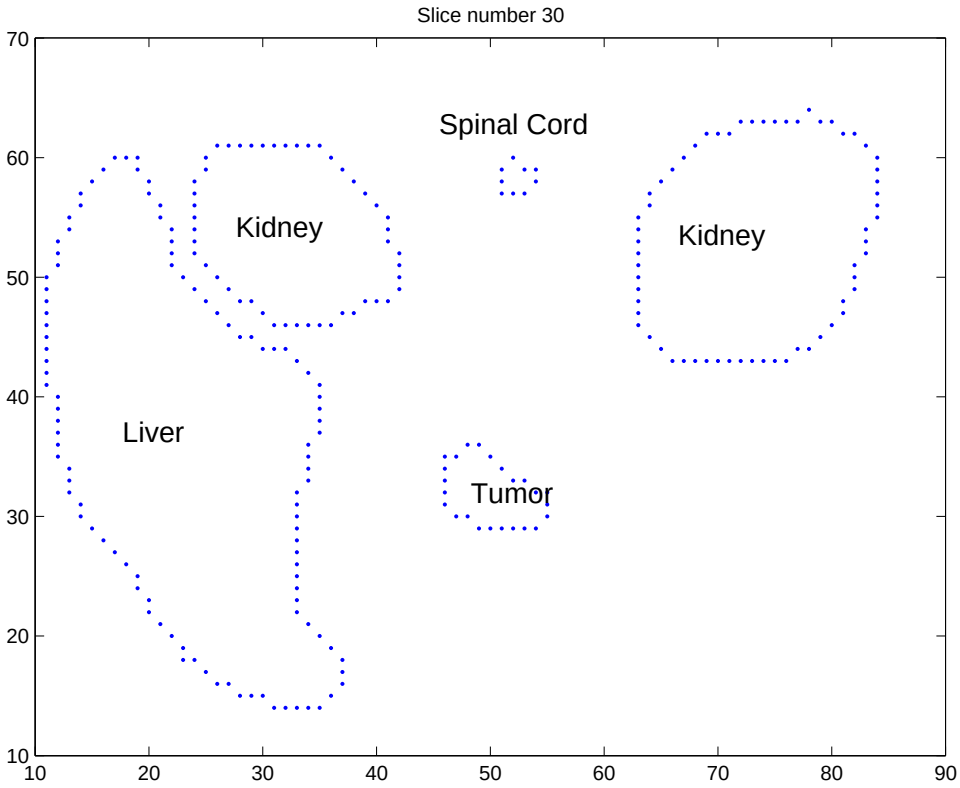


Figure 18: Patient planning problem.

with no dose delivered to the sensitive or normal tissues:

$$T^*(i,j,k) = \begin{cases} 1 & \text{if } (i,j,k) \in \mathcal{T} \\ 0 & \text{if } (i,j,k) \in \mathcal{S} \cup \mathcal{N}. \end{cases} \tag{4.6}$$

However, this is not possible as radiation beams must pass through normal and/or sensitive tissues to reach the tumor. (See [65] for a description.) Instead, application experts allow for some error by reducing their requirements to the following [65]:

$$T(\mathcal{T}) \in [0.95, 1.07], \tag{4.7}$$

$$90\% \text{ of sensitive tissues should have } T(\mathcal{S}) \leq 0.2 \cdot T^*(\mathcal{T}) = 0.2, \tag{4.8}$$

and

$$T(\bar{\mathcal{N}}) \text{ should be as small as possible} \tag{4.9}$$

where $\bar{\mathcal{N}}$ is a reduced set of normal tissue voxels, consisting of those normal tissue voxels around the tumor and a sampling of the other normal tissue voxels. In our case, $\bar{\mathcal{N}}$ consists of $96,154$ voxels.

4.5.1 Simple Shifts

Initially, we consider a direct translation from the simple example. As before, we allow for a shift of one or two voxels. In the three-dimensional example,

we allow for six possible shift directions (up, down, left, right, forward, back); this gives thirteen total possibilities (including no shift). Here again, we consider three volatilities, based on the volatilities from the one-dimensional case: low volatility (probability of a shift is 20%), medium volatility (probability of a shift is 40%), and high volatility (probability of a shift is 60%). In addition, we consider a very volatile case, where the probability of a shift is 0.78. We also assume that the ideal dose can be delivered, and so we measure the error against T^* from equation (4.6). Table 17 displays the average results for the constant, reactive and NDP simple rule of thumb policies after 1000 simulations. Displayed are the errors on the tumor, the errors on the sensitive structures, the errors on the normal tissues, and the overall error. The overall error is a linear combination of the three other errors, found by weighting the areas as in the one-dimensional examples: the weight on the tumor is 10, the weight on the sensitive structures is 5, and the weight on the normal tissues is 1.

The table shows that the simple NDP policy significantly outperforms the constant policy on the tumor in all cases. It also does very well compared to the reactive policy, beating the reactive policy in all cases except for the very high volatility. This is not surprising since the NDP policy was built from considering only the low, medium and high volatilities, but not the very high volatility. Thus, the NDP policy should be modified for very high volatility cases. Examining the sensitive and normal errors for this case, we see that

Policy - Volatility	$\mathcal{T}$ Error	$\mathcal{S}$ Error	$\mathcal{N}$ Error	Overall Error
Constant - Low	54.3	0.5	53.8	599.3
Reactive - Low	5.1	0.6	58.4	112.4
Simple NDP - Low	0.3	0.6	60.1	66.1
Constant - Medium	107.7	1.0	106.7	1188.7
Reactive - Medium	18.7	1.1	127.6	320.1
Simple NDP - Medium	5.0	1.2	134.6	190.6
Constant - High	151.7	1.0	150.7	1672.7
Reactive - High	48.5	1.4	201.5	693.5
Simple NDP - High	34.3	1.4	221.1	571.1
Constant - Very High	263.1	4.7	258.4	2912.9
Reactive - Very High	145.6	6.7	403.4	1892.9
Simple NDP - Very High	166.5	7.4	481.9	2183.9

Table 17: Results of simple shifts on the real-life example.

the NDP policy's error is higher than the reactive's error. This suggests that the NDP policy is too aggressive under the very high volatility.

Under the other volatilities, the NDP policy has about the same error as the reactive policy on the sensitive tissue, and slightly worse error on the normal tissue. These results are not surprising, considering the simple one-dimensional examples that were used to build the NDP policy. We note that the simple NDP policy was built focusing on the tumor: in the one-dimensional examples, the tumor areas (spikes) were always weighted high. As a result, the NDP policy concentrates on fully dosing the tumor, and attempts to avoid the normal and sensitive tissues as a secondary consideration. Thus, we expect that the error on the tumor is low, while the error on the normal and sensitive tissues could be higher.

Also, we note that the simple NDP policy delivers more overall dose to the patient than the constant policy. In addition, most of this dose is delivered early on, when the residual is high (since we deliver 0.4 of the high residual). Due to the shifts, this higher dose is generally not delivered to the tumor (although some of the tumor may be overdosed). Thus, we would expect that the error on the normal and sensitive tissues to be larger for the NDP policy.

Like the NDP policy, the reactive policy may deliver more dose overall than the constant policy. However, in the ideal case of no shifts, the reactive policy reduces exactly to the constant policy. Unlike the NDP policy, where

a great deal of dose is delivered early on, the reactive policy policy starts out by delivering little dose early on (1/20-th of the residual — exactly the constant policy at the first time stage). Thus, at the last time stage, where we deliver the full residual, the reactive policy may be delivering a large amount of dose to make up for errors earlier on. As a result, a shift in the last policy can be disastrous for the reactive policy.

As noted earlier, unlike the NDP or the reactive policies, the constant policy delivers exactly the same dose at every time stage. Because it does not adjust its dose for errors on the tumor, the constant policy tends to underdose the tumor. In fact, as can be seen from Table 17, the constant policy error on the tumor is the sum of the errors on the normal and sensitive tissues — the tumor dose was simply shifted to the normal and sensitive tissues instead.

4.5.2 Realistic Shifts

In the previous section, we only allowed for a maximum of two shifts. In this section, we consider much larger shifts, in order to consider more a more realistic example. To determine the shifts that occur, we first choose the direction of the shift — as before, we have six directions (w_k^1 is up, down, back, forward, left, right), plus one for no shift. We consider the same four volatilities (low, medium, high, very high) in choosing the directions.

After the direction is determined, the size of the shift is chosen. We allow

for 18 possible shift sizes, varying from 1 to 18, with decreasing probability:

$$w_k^2 = \begin{cases} 1 \text{ with probability } 0.25; & 2 \text{ with probability } 0.18 \\ 3 \text{ with probability } 0.12; & 4 \text{ with probability } 0.10 \\ 5 \text{ with probability } 0.08; & 6 \text{ with probability } 0.05 \\ 7 \text{ with probability } 0.04; & 8 \text{ with probability } 0.032 \\ 9 \text{ with probability } 0.027; & 10 \text{ with probability } 0.024 \\ 11 \text{ with probability } 0.022; & 12 \text{ with probability } 0.019 \\ 13 \text{ with probability } 0.016; & 14 \text{ with probability } 0.014 \\ 15 \text{ with probability } 0.011; & 16 \text{ with probability } 0.008 \\ 17 \text{ with probability } 0.005; & 18 \text{ with probability } 0.002 \end{cases}$$

Again, we assume that the ideal dose can be delivered, and so compute the error with respect to T^* from equation (4.6). Table 18 shows the average results for the constant, reactive, and NDP policies after 1000 simulations.

As Table 18 shows, including larger shifts does not hurt the simple NDP policy, but rather helps it. In all cases (including the very volatile case), the simple NDP policy significantly outperforms the constant and reactive policies on the tumor. Here again, the errors on the sensitive and normal tissues are generally (around 15%) higher for the NDP policy than for either

Policy - Volatility	$\mathcal{T}$ Error	$\mathcal{S}$ Error	$\mathcal{N}$ Error	Overall Error
Constant - Low	120.3	6.2	114.1	1348.1
Reactive - Low	8.6	6.8	126.5	246.5
Simple NDP - Low	0.138	7.1	124.5	161.38
Constant - Medium	235.6	12.2	223.4	2640.4
Reactive - Medium	29.1	15.2	276.3	643.3
Simple NDP - Medium	2.5	16.2	272.9	378.9
Constant - High	353.5	18.5	335.0	3962.5
Reactive - High	68.3	25.6	467.3	1278.3
Simple NDP - High	18.4	27.5	486.7	808.2
Constant - Very High	461.3	24.1	437.3	5170.8
Reactive - Very High	138.7	37.1	690.7	2263.2
Simple NDP - Very High	88.3	42.8	770.3	1867.3

Table 18: Results of realistic shifts on the real-life example.

the constant and reactive, although the NDP policy error is typically close to the reactive error.

4.5.3 Replanning

In the previous examples, we assumed that we could achieve any dose that was necessary for the policies. However, these doses may not be physically implementable. To obtain an implementable dose, we must use a planning tool. Our work thus far has been independent of any planning tools. In this section, we make use of a particular planning tool, presented in [65], to demonstrate how the policies perform under actual replanning.

Given the ideal dose T^* from equation (4.6), the planning tool in [65] solves a mixed-integer programming problem to determine the angles (out of a total of 36) from which individual beams of radiation should be delivered and the length of time that each delivery should last. To reduce the solution time, we preselect ten angles to use. This reduces the problem to a linear

programming problem:

$$\begin{aligned}
\min_{w,D} \quad & \lambda_t(\|(D_{\mathcal{T}} - \theta_L(\mathcal{T}))_+\|_\infty + \|(\theta_U(\mathcal{T}) - D_{\mathcal{T}})_+\|_\infty) \\
& + \tfrac{\lambda_s}{C_s}\|(D_{\mathcal{S}} - \phi)_+\|_1 + \tfrac{\lambda_n}{C_n}\|D_{\mathcal{N}}\|_1 \\
\text{subject to} \quad & D_\Omega = \sum_{A \in \mathcal{A}} \text{Dose}(A, \Omega) \cdot w_A, \quad \Omega = \mathcal{T} \cup \mathcal{S} \cup \mathcal{N} \\
& D_{\mathcal{T}} \leq u \\
& 0 \leq w_A \leq M, \qquad\qquad \forall A \in \mathcal{A}
\end{aligned} \tag{4.10}$$

Here, $\mathcal{A}$ is the set of (ten) angles that can be used to deliver radiation. w_A represents the length of radiation exposure time for each angle $A \in \mathcal{A}$, bounded above by M (M is predetermined by application requirements). $\text{Dose}(A, \Omega)$ is a data matrix that contains the amount of radiation that is delivered to each $(i, j, k) \in \Omega = \mathcal{T} \cup \mathcal{S} \cup \mathcal{N}$ when $w_A = 1$. D_Ω is the total radiation dose delivered to each voxel in Ω. Note that $D_{\mathcal{T}}$ is bounded above by u; we set

$$u = 1.15 \cdot \theta(i, j, k) \;\; \forall (i, j, k) \in \mathcal{T},$$

where $\theta(i, j, k)$ is the desired dose at voxel (i, j, k). Here, we do not assume a general upper bound on the amount of dose that can be delivered to each voxel, choosing instead to limit the dose based on the desired dose. However, u could also be used to limit the total dose delivered during each stage: for example, we could set $u = 1/10$ (twice the ideal constant dose) to correspond to the upper bound from the simple one-dimensional examples. θ_L and θ_U

are the lower and upper bounds, respectively, on the acceptable tumor dose. From the prescription (4.7),

$$\theta_L(i,j,k) = 0.95 \cdot \theta(i,j,k) \;\; \forall (i,j,k) \in \mathcal{T} \tag{4.11}$$

and

$$\theta_U(i,j,k) = 1.07 \cdot \theta(i,j,k) \;\; \forall (i,j,k) \in \mathcal{T}. \tag{4.12}$$

ϕ is the acceptable upper bound on the sensitive tissue. From the prescription (4.8),

$$\phi = 0.2 \cdot (\text{fraction of total dose to be delivered}).$$

(Note that, due to the difficulty in applying this restriction to only 90% of the sensitive tissue, as in the original prescription, we instead apply it to all of the sensitive tissue.) λ_t, λ_s, and λ_n are parameters that weigh the importance of the prescriptions for the tumor, sensitive structures, and normal tissues, respectively. We set $\lambda_t = \lambda_s = \lambda_n = 10$. C_s and C_n represent the cardinality of the sensitive and normal tissues, respectively, allowing us to compare the average error on the sensitive and normal tissues to the maximum error on the tumor. In order to perform replanning, we must provide model (4.10) with appropriate θ and ϕ values.

To apply the constant policy, we need only solve model (4.10) once. If we solve model (4.10) with

$$\theta(i,j,k) = 1 \;\; \forall (i,j,k) \in \mathcal{T}$$

and

$$\phi(i,j,k) = 0.2 \;\; \forall (i,j,k) \in \mathcal{S},$$

we find the implementable dose for all of T^*. Then, we can divide the resulting dose by 20 to obtain the implementable dose to deliver at each time stage. (Note that each time stage's dose is implementable: we essentially use $w_A/20$.)

To apply the reactive policy, we must replan at each time stage. This requires 20 solutions to model (4.10), each with a different θ and ϕ. For time stage k, we set

$$\theta(i,j,k) = \max\left\{0, \frac{T^*(i,j,k) - x_k(i,j,k)}{N-k}\right\} \;\; \forall (i,j,k) \in \mathcal{T}$$

and

$$\phi(i,j,k) = 0.2/(N-k) \;\; \forall (i,j,k) \in \mathcal{S}.$$

Like the reactive policy, the NDP policy requires 20 solutions to model (4.10). To obtain θ, we use the simple rule categorical multipliers from Table 15: for (i,j,k) in category c (= low, medium, or high residual),

$$\theta(i,j,k) = m_k(c) \cdot (T^*(i,j,k) - x_k(i,j,k))$$

where $m_k(c)$ is the k-th multiple for category c. ϕ is similar to that for reactive, except we conform to the prescription and do not restrict a percentage

of the sensitive tissue in each structure:

$$\phi(i,j,k) = \begin{cases} 0.2 \cdot m_k(\text{low}) & \text{for the smallest } 0.90 \cdot m_k(\text{low}) \text{ of each structure} \\ \infty & \text{for the largest } 0.10 \cdot m_k(\text{low}) \text{ of each structure.} \end{cases}$$

Note that the replanning that is done for the reactive and simple NDP policies is another example of slice modeling: θ and ϕ are only data to the replanning model (4.10). The slice modeling approach could be advantageous in these cases, particularly since it would allow us to keep the very large Dose matrix in solver memory between solves. However, we cannot use the slice interface described in Chapter 2 here since we do not know θ and ϕ for all problems initially. θ and ϕ are only defined once the previous time stage has been completed. This would require execution time reading of the data. Under GDX functions [81], the slice interface may be able to handle these types of problems in the future.

We apply the realistic shifts from the previous section, under the very high probability (probability of a shift is 0.78). Table 19 shows the results for the constant, reactive and simple NDP policies after 20 simulations. The errors given correspond to errors on the prescriptions (4.7), (4.8) and (4.9).

As shown in Table 19, the reactive and NDP policies significantly outperform the constant policy on the tumor, cutting the error by more than one-sixth. However, the error on the sensitive and normal tissues is also significantly higher — around three times more on the normal tissue, and a great

Policy	$\mathcal{T}$ Error	$\mathcal{S}$ Error	$\mathcal{N}$ Error	Overall Error
Constant	613.5	0.0	2743.9	8878.9
Reactive	41.2	242.6	6633.0	8258.0
Simple NDP	75.2	62.3	6454.5	7518.0

Table 19: Results of replanning on the real-life example under very high probability.

deal more on the sensitive structures. Note also that the reactive policy does better than the NDP policy on the tumor, but the reactive's overall error is worse. This comes from the fact that the NDP policy performs significantly better on the sensitive structures, cutting that error by about one-fourth.

To see how well the different policies treat the patient, we can also consider their average dose volume histograms (Figures 19, 20, and 21). These figures compare the relative dose delivered to the fraction of volume of the structure (either the tumor, sensitive structures, or normal tissue) that received the dose. Under the tumor prescription (4.7), we would like the tumor curve to be flat across the top of the plot and then drop to zero between 0.95 and 1.07. As can be seen from the three figures, none of the policies achieve this. The constant policy is the worst, delivering only approximately 0.5 of the full dose to the complete tumor. The reactive policy is significantly better, delivering approximately 0.7 of the full dose to the complete tumor. The simple NDP policy is not as good as the reactive policy on the tumor

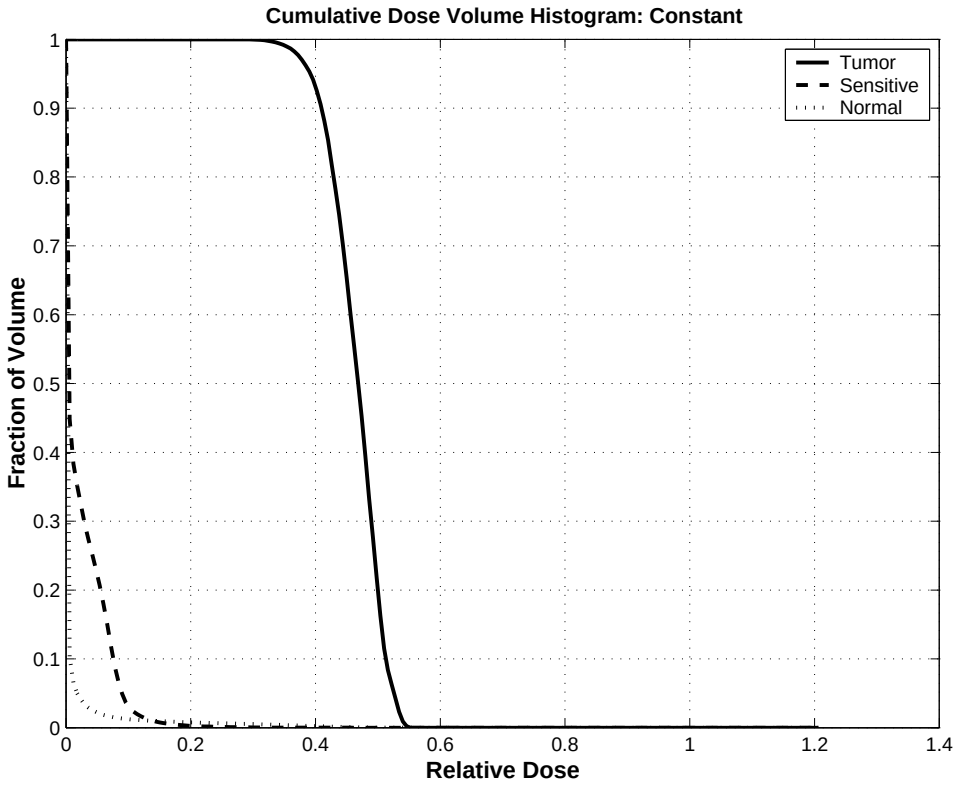


Figure 19: Dose Volume Histogram for the Constant Policy

(as we saw under the errors in Table 19), however it is better than the constant: although it delivers about the same amount of dose to the complete tumor, it does deliver more dose to parts of the tumor (about 70% of the tumor receives 90% of the dose). We can see that the NDP policy, though, performs better on the sensitive structures than the reactive policy.

Considering Table 18, we would have expected the simple NDP policy to outperform both the constant and the reactive policy. However, this is not the case. This suggests that the NDP policy suffers more from replanning

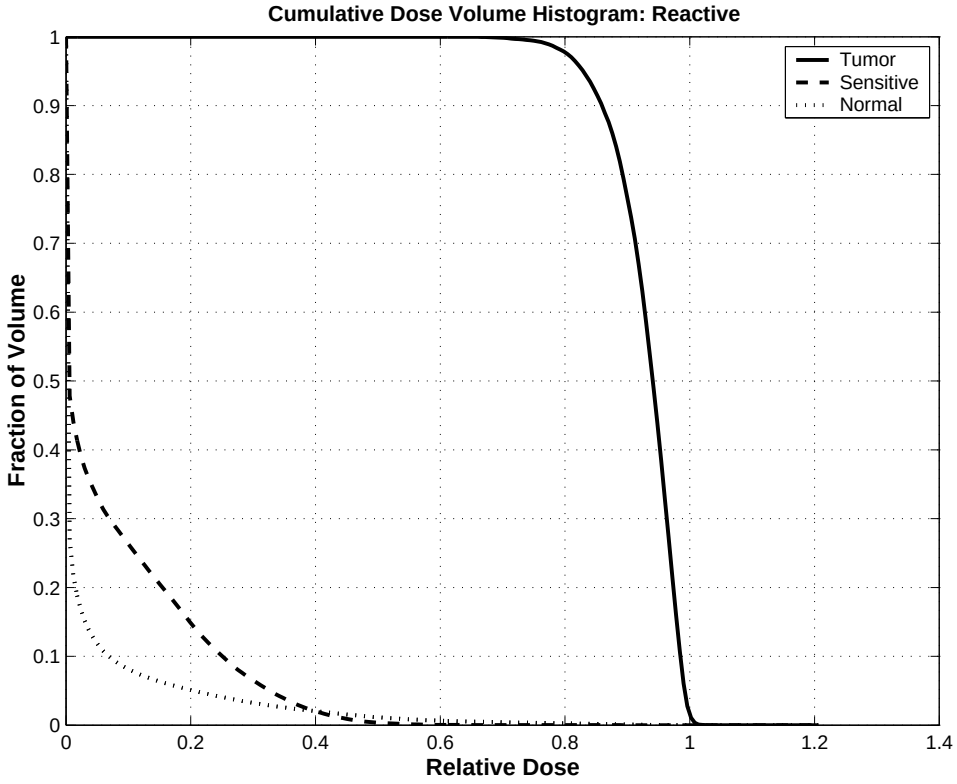


Figure 20: Dose Volume Histogram for the Reactive Policy

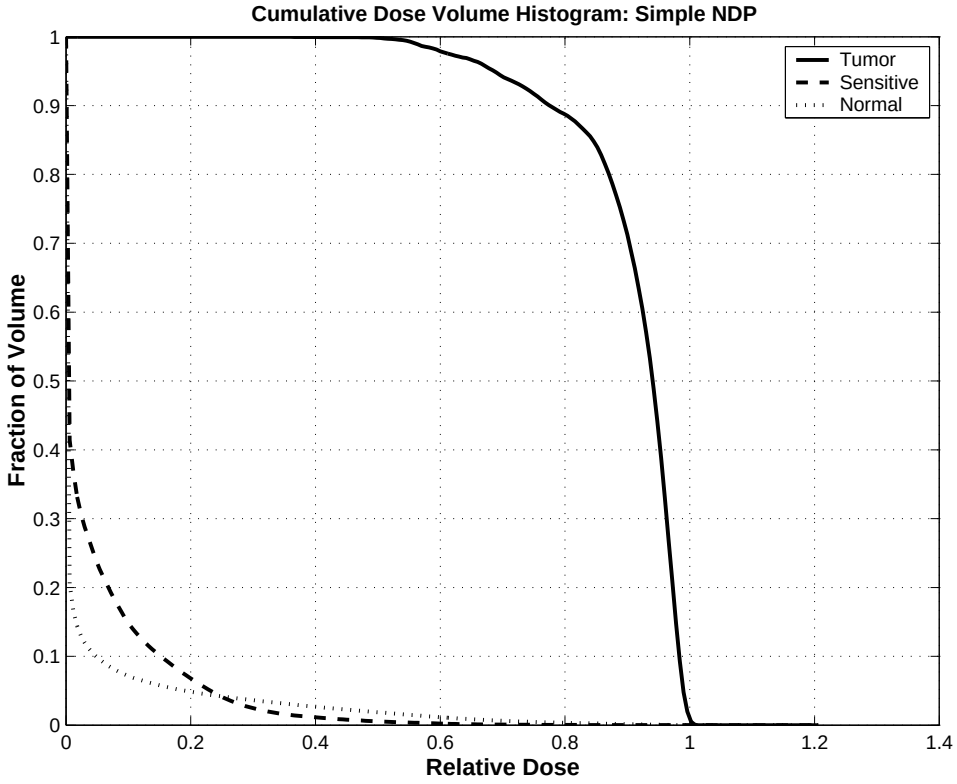


Figure 21: Dose Volume Histogram for the Simple NDP Policy

error than the reactive policy, further suggesting that the reactive policy is easier to plan. Support for this conclusion comes from the NDP dose volume histogram (Figure 21): under replanning, the NDP policy underdoses the tumor. Since the NDP policy requests that more dose be delivered (depending upon the residual, but particularly initially), this underdosing suggests that the planning problem (4.10) delivers less actual dose to the tumor. Considering a more advanced planning problem may improve the NDP results. In addition, incorporating a "replanning error" term in the simulation during the building of the NDP policies may yield more robust simple NDP policies.

For immediate use, we suggest that either the reactive policy or the NDP simulation approach be employed, as long as the errors on the sensitive and normal tissues are acceptable. If these errors are too high, a different weighting scheme that puts more emphasis on avoiding the sensitive and/or normal tissues can be used. The process though, remains the same. On a day-to-day basis, the treatment planner, knowing x_k, can calculate the dose required at each voxel in the manner outlined in Section 4.2.4, accounting for the stochastic errors that have occurred. Once this dose distribution is known, existing planning tools can be used to implement this on particular machines, as we have demonstrated here.

Chapter 5

Conclusions

In this work, we considered systems whose solutions depend upon mathematical programming problems. By examining how the programs were integrated into their respective systems, we demonstrated improvements to example systems. Three specific systems were considered: slice modeling, model building and treatment planning. All three systems were data-intensive and required multiple solutions to mathematical programming problems.

We began by discussing slice models, which are found in various systems (including the other systems that we considered). Using a slice model formulation, we showed how defining individual programs from the model by data slices suggests a new solution technique. By implementing this technique in the slice interface, we were able to take advantage of the fact that program structure and core data stay the same between programs. This led to faster solution times, as we demonstrated on real-world examples from both DEA and cross-validation. In addition, under the improvements we made to the DEA models, we were able to extend the results of DEA problems to include confidence intervals by the application of a computationally-intensive

bootstrap technique.

Because the slice interface was built for a general modeling language, we are able to solve very different slice models. We focused on DEA and cross-validation models, but nothing in the slice interface limits us to only these types of problems. Any linear, mixed integer or simple quadratic slice model can make use of the slice interface, as long as the data for all of the solves is available at the start. Although this restriction prevents us from currently employing the slice interface on all slice models (such as the replanning model from Section 4.5), we are still able to apply it to a number of other systems. For example, determining the effects of particular scenarios on a robust solution in the framework of stochastic optimization can be formulated as a slice model and solved under the slice interface (provided the scenarios are made available in advance for the initial model generation). This allows us to extend the efficiency we achieve under DEA to other areas.

Another system we considered was model building. We discussed this system by considering a specific approach called likelihood basis pursuit (LBP). Under LBP, a model for predicting the probability of a medical outcome is obtained. This is done by using a smoothing spline ANOVA model to decompose the log odds ratio into parametric and smooth components. Weights on the various components are determined by maximizing a penalized likelihood function.

As we discussed, the final LBP model can only be obtained after appropriate penalty parameters are found. We required penalty parameters that minimized the misclassification error for the current problem. To find these parameters, we initially considered using a grid search. Under the grid search, the search for penalty parameters resulted in a slice model, and slice modeling techniques were employed (although the slice interface was not used since it does not handle general nonlinear models). We also showed that the penalty parameter search could be improved, both by being made more efficient (considering the number of calculations for misclassification) and by finding better points, through the use of derivative-free optimization techniques. These techniques could be implemented to employ some of the slice modeling techniques, namely maintaining program structure and starting later solves from earlier solutions, as they also require multiple solutions to data-modified programs.

The last system we considered was that of day-to-day radiation treatment planning. We developed a simple stochastic linear program to model the day-to-day planning problem and showed how, even on the simple model, dynamic programming was intractable for realistic time stages. To obtain approximate solutions, we turned to neuro-dynamic programming coupled with heuristic policies. This allowed us to determine good policies that outperformed the (currently-used) constant policy. Further, this improvement was maintained when target-structure dependence was removed. This allowed us to define a

simple rule of thumb, useful for off-line planning.

All of this was done very generally, without considering specific planning tools or delivery mechanisms. To see if our improvements would still be maintained in specific planning examples, we applied the simple rule of thumb to actual patient data. To perform the replanning necessary at every stage, we employed the planning tool from [65], which makes use of optimization to find an implementable dose. Under this planning tool, we could view the treatment planning system as a slice model. Under replanning, the reactive heuristic outperformed the simple rule of thumb on the tumor, although results under perfect planning suggest that the simple rule would be improved with better replanning. Further, we found significant improvement over the currently-used constant policy.

These systems showed the benefits of considering how optimization is integrated into systems. By considering mathematical programs that need to be solved and using techniques to improve these programs, we were able to significantly improve the efficiency of the entire system. This allowed larger and more complicated examples to be solved and extended. Further, better solutions were obtained by looking at how the optimization could be improved. Alternative optimization methods yielded solutions for realistic instances in some cases, and improved solutions in others. By improving the optimization, the entire system benefited.

Bibliography

[1] Agha Iqbal Ali and Lawrence M. Seiford. Computational accuracy and infinitesimals in Data Envelopment Analysis. *INFOR*, 31(4):290–297, November 1993.

[2] Agha Iqbal Ali and Lawrence M. Seiford. The mathematical programming approach to efficiency analysis. In Harold O. Fried, C.A. Knox, and Shelton S. Schmidt, editors, *The Measurement of Productive Efficiency: Techniques and Applications*, chapter 3, pages 120–159. Oxford University Press, 1993.

[3] R. Allen, A. Athanassopoulos, R. G. Dyson, and E. Thanassoulis. Weights restrictions and value judgements in Data Envelopment Analysis: Evolution, development and future directions. *Annals of Operations Research*, 73:13–34, 1997.

[4] Per Andersen and Niels Christian Petersen. A procedure for ranking efficient units in Data Envelopment Analysis. *Management Science*, 39(10):1261–1264, October 1993.

[5] Tim Anderson. A Data Envelopment Analysis (DEA) home page. HTML document, June 1996. `http://www.emp.pdx.edu/dea/homedea.html`.

[6] R. D. Banker, A. Charnes, and W.W. Cooper. Some models for estimating technical and scale inefficiencies in Data Envelopment Analysis. *Management Science*, 30(9):1078–1092, September 1984.

[7] Rajiv D. Banker and Richard C. Morey. Efficiency analysis for exogenously fixed inputs and outputs. *Operations Research*, 34(4):513–521, July–August 1986.

[8] Rajiv D. Banker and Richard C. Morey. The use of categorical variables in Data Envelopment Analysis. *Management Science*, 32(12):1613–1627, December 1986.

[9] K. P. Bennett and C. Campbell. Support vector machines: Hype or hallelujah? *SIGKDD Explorations*, 2(2), 2000.

[10] Dimitri P. Bertsekas. Differential training of rollout policies. In *Proceedings of the 35th Allerton Conference on Communication, Control, and Computing*, 1997. Available as PDF document from `http://www.mit.edu:8001//people/dmitrib/Diftrain.pdf`.

[11] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, Massachusetts, 1996.

[12] J. R. Birge and R. Louveaux. *Introduction to Stochastic Programming*. Springer, New York, 1997.

[13] T. Bortfeld. Current status of IMRT: physical and technological aspects. *Radiotherapy and Oncology*, 61(2):291–304, 2001.

[14] T. Bortfeld and W. Schlegel. Optimization of beam orientations in radiation therapy: some theoretical considerations. *Physics in Medicine and Biology*, 38(2):291–304, 1993.

[15] P. S. Bradley and O. L. Mangasarian. Feature selection via concave minimization and support vector machines. In J. Shavlik, editor, *Machine Learning Proceedings of the Fifteenth International Conference (ICML '98)*, pages 82–90, San Francisco, California, 1998. Morgan Kaufmann.

[16] A. Brahme. Treatment optimization: Using physical and radiobiological objective functions. In A. R. Smith, editor, *Radiation Therapy Physics*, pages 209–246. Springer-Verlag, Berlin, 1995.

[17] Anthony Brooke, David Kendrick, Alexander Meeraus, and Ramesh Raman. GAMS: A user's guide. PDF document, December 2002. `http://www.gams.com/docs/document.htm`.

[18] A. Charnes and W. W. Cooper. Programming with linear fractional functionals. *Naval Research Logistics Quarterly*, 9:181–185, 1962.

[19] A. Charnes, W. W. Cooper, and E. Rhodes. Measuring the efficiency

of decision making units. *European Journal of Operational Research*, 2:429–444, 1978.

[20] A. Charnes, W. W. Cooper, and E. Rhodes. Short communication: Measuring the efficiency of decision-making units. *European Journal of Operations Research*, 3:339, 1979.

[21] A. Charnes, W. W. Cooper, and E. Rhodes. Evaluating program and managerial efficiency: An application of Data Envelopment Analysis to Program Follow Through. *Management Science*, 27(6):668–697, June 1981.

[22] A. Charnes, S. Haag, P. Jaska, and J. Semple. Sensitivity of efficiency classifications in the additive model of data envelopment analysis. *International Journal of Systems Sciences*, 23(5):789–798, 1992.

[23] A. Charnes and L. Neralic. Sensitivity analysis of the additive model in data envelopment analysis. *European Journal of Operational Research*, 48:332–341, 1990.

[24] Abraham Charnes, William Cooper, Arie Y. Lewin, and Lawrence M. Seiford. *Data Envelopment Analysis: Theory, Methodology and Applications*. Kluwer Academic Publishers, Boston, 1994.

[25] Scott Shaobing Chen, David L. Donoho, and Michael A. Saunders.

Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61, 1998.

[26] P. S. Cho, S. Lee, R. J. Marks, S. Oh, S. Sutlief, and H. Phillips. Optimization of intensity modulated beams with volume constraints using two methods: cost function minimization and projection onto convex sets. *Medical Physics*, 25(4):435–443, 1998.

[27] Vašek Chvátal. *Linear Programming.* W.H. Freeman and Company, 1983.

[28] William W. Cooper, Lawrence M. Seiford, and Kaoru Tone. *Data Envelopment Analysis: A Comprehensive Text with Models, Applications, References and DEA-Solver Software.* Kluwer Academic Publishers, Boston, 2000.

[29] D. M. Dolan, A. H. El-Shaarawi, and T. B. Reynoldson. Predicting benthic counts in Lake Huron using spatial statistics and quasi-likelihood. *Environmetrics*, 11(3):287–304, 2000.

[30] R. G. Dyson, E. Thanassoulis, and A. Boussofiane. A DEA (Data Envelopment Analysis) tutorial. HTML document, July 1995. `http://www.warwick.ac.uk/~bsrnb/pages/links/dea.htm`.

[31] Bradley Efron. *The Jackknife, the Bootstrap and Other Resampling*

Plans, volume 38 of *CMBS-NSF Regional Conference Series in Applied Mathematics*. SIAM, Philadelphia, Pennsylvania, 1982.

[32] Bradley Efron and Robert J. Tibshirani. *Introduction to the Bootstrap*. Chapman and Hall, New York, 1993.

[33] R. Ejrnaes, E. Aude, B. Nygaard, and B. Munier. Prediction of habitat quality using ordination and neural networks. *Ecological Applications*, 12(4):1180–1187, August 2002.

[34] M. J. Farrell. The measurement of productive efficiency. *Journal of the Royal Statistical Society, Series A (General)*, 120(3):253–290, 1957.

[35] M. C. Ferris, J.-H. Lim, and D. M. Shepard. Optimization approaches for treatment planning on a Gamma Knife. *SIAM Journal on Optimization, forthcoming*, 2002.

[36] M. C. Ferris, J.-H. Lim, and D. M. Shepard. Radiosurgery treatment planning via nonlinear programming. *Annals of Operations Research, forthcoming*, 2002.

[37] M. C. Ferris and T. S. Munson. Interior point methods for massive support vector machines. Technical Report 00-05, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, WI, 2000.

[38] M. C. Ferris and M. M. Voelker. Slice models in general purpose modeling systems: An application to DEA. *Optimization Methods and Software*, 2002. forthcoming.

[39] E. E. Fitchard, J. S. Aldridge, P. J. Reckwerdt, and T. R. Mackie. Registration of synthetic tomographic projection data sets using cross-correlation. *Physics in Medicine and Biology*, 43:1645–1657, 1998.

[40] GAMS Development Corporation. Data Envelopment Analysis — DEA (DEA,SEQ=192). GAMS model. Obtained from the GAMS library call 'gamslib dea' (September 1999).

[41] GAMS Development Corporation. Slice modeling in GAMS. HTML document. `http://www.gams.com/contrib/gamsdea/dea.htm`.

[42] GAMS Development Corporation. Quadratic programs in GAMS. HTML document, 2001. `http://www.gams.com/contrib/qpwrap/qpwrap.htm`.

[43] Alexander Gocht. Private communication via email, August 2002.

[44] H. W. Hamacher and K.-H. Küfer. Inverse radiation therapy planning — a multiple objective optimisation approach. Technical Report 12, Institute for Techno- and Econo-Mathematics (ITWM) and Department of Mathematics, University of Kaiserslautern, Kaiserslautern, Germany, 1999. Available as PDF document

from http://www.itwm.uni-kl.de/zentral/download/berichte/bericht12.pdf.

[45] J. C. Hargreaves and J. D. Annan. Assimilation of paleo-data in a simple earth system model. *Climate Dynamics*, 19:371–381, August 2002.

[46] H. Heitsch and W. Römisch. Scenario reduction algorithms in stochastic programming. Preprint 01-8, Institut für Mathematik, Humboldt-Universität, Berlin, 2001.

[47] T. Holmes and T. R. Mackie. A filtered backprojection dose calculation method for inverse treatment planning. *Medical Physics*, 21(2):303–313, 1994.

[48] ILOG CPLEX Division, 889 Alder Avenue, Incline Village, Nevada. *CPLEX Optimizer*. http://www.cplex.com/.

[49] International Commison on Radiation Units and Measurments, Inc. Prescribing, recording, and reporting photon beam therapy. ICRU Report 50, 1993.

[50] International Commison on Radiation Units and Measurments, Inc. ICRU report 62: Prescribing, recording and reporting photon beam therapy (supplement to ICRU report 50). *ICRU News*, December

1999. Available as HTML document from `http://www.icru.org/n_992_4.htm`.

[51] International Commison on Radiation Units and Measurments, Inc. Prescribing, recording, and reporting photon beam therapy (supplement to ICRU report 50). ICRU Report 62, 1999.

[52] L. Johansen. Production functions and the concept of capacity. In *Recherches recentes sur la fonction de production*. Namur: Centre d'Etudes et de la Recherche Universitaire de Namur, 1968.

[53] Gordon Johnston. SAS software to fit the generalized linear model. In *SUGI Proceedings*, 1993. Available as PDF document from `http://www.sas.com/rnd/app/papers/abstracts/genmod.html`.

[54] P. Kall and S. W. Wallace. *Stochastic Programming*. John Wiley & Sons, Chichester, 1994.

[55] Wagner A. Kamakura. A note on "The use of categorical variables in Data Envelopment Analysis". *Management Science*, 34(10):1273–1276, October 1988.

[56] P. J. Keall, V. R. Kini, S. S. Vedam, and R. Mohan. Motion adaptive x-ray therapy: a feasibility study. *Physics in Medicine and Biology*, 46(1):1–10, January 2001.

[57] S. Keles, M. van der Laan, and M. B. Eisen. Identification of regulatory elements using a feature selection method. *Bioinformatics*, 18(9):1167–1175, September 2002.

[58] Hideo D. Kubo and Bruce C. Hill. Respiration gated radiotherapy treatment: a technical study. *Physics in Medicine and Biology*, 41(1):83–91, January 1996.

[59] K.-H. Küfer, H. W. Hamacher, and T. Bortfeld. A multicriteria optimization approach for inverse radiotherapy planning. Symposium on Operations Research (OR 2000). Available as PDF document from `http://www.uni-duisburg.de/FB5/BWL/WI/or2000/sektion01/kuefer.pdf`.

[60] Ludwig Kuntz and Stefan Scholtes. Measuring the robustness of empirical efficiency valuations. *Management Science*, 46(6):807–823, June 2000.

[61] Jeffrey C. Lagarias, James A. Reeds, Margaret H. Wright, and Paul E. Wright. Convergence properties of the Nelder-Mead simplex method in low dimensions. *SIAM Journal on Optimization*, 9(1):112–147, 1998.

[62] Y. J. Lee and O. L. Mangasarian. SSVM: A smooth support vector machine for classification. *Computational Optimization and Applications*, 20(1):5–22, October 2001.

[63] A. Y. Lewin and L. M. Seiford. Extending the frontiers of Data Envelopment Analysis. *Annals of Operations Research*, 73:1–11, 1997.

[64] Jonathan G. Li and Lei Xing. Inverse planning incorporating organ motion. *Medical Physics*, 27(7):1573–1578, July 2000.

[65] Jinho Lim, Michael C. Ferris, Stephen J. Wright, David M. Shepard, and Matthew A. Earl. An optimization framework for conformal radiation therapy. Technical report, Computer Sciences Department, University of Wisconsin, Madison, WI, 2002.

[66] Xiwu Lin, Grace Wahba, Dong Xiang, Fangyu Gao, Ronald Klein, and Barbara Klein. Smoothing spline ANOVA models for large data sets with bernoulli observations and the randomized GACV. *Annals of Statistics*, 28(6):1570–1600, 2000.

[67] Yi Lin, Grace Wahba, Hao Zhang, and Yoonkyung Lee. Statistical properties and adaptive tuning of support vector machines. Technical Report 1022, Department of Statistics, University of Wisconsin, Madison, WI, September 2000. Available as postscript document from `ftp://ftp.stat.wisc.edu/pub/wahba/tr1022.ps`.

[68] B. K. Lind, P. Kǎllman, B. Sundelin, and A. Brahme. Optimal radiation beam profiles considering uncertainties in beam patient alignment. *Acta Oncologica*, 32:331–342, 1993.

[69] J. T. Linderoth, A. Shapiro, and S. J. Wright. The empirical behavior of sampling methods for stochastic programming. Optimization Technical Report 02-01, Computer Science Department, University of Wisconsin, Madison, Wisconsin, 2002.

[70] M. Livny. The Condor project homepage. `http://www.cs.wisc.edu/condor`.

[71] Johan Löf, Bengt K. Lind, and Anders Brahme. Optimal radiation beam profiles considering the stochastic process of patient positioning in fractionated radiation therapy. *Inverse Problems*, 11:1189–1209, 1995.

[72] Johan Löf, Bengt K. Lind, and Anders Brahme. An adaptive control algorithm for optimization of intensity modulated radiotherapy considering uncertainties in beam profiles, patient set-up and internal organ motion. *Physics in Medicine and Biology*, 43:1605–1628, 1998.

[73] T. R. Mackie, T. Holmes, S. Swerdloff, P. Reckwerdt, J. O. Deasy, J. Yang, B. Paliwal, and T. Kinsella. Tomotherapy: a new concept for the delivery of dynamic conformal radiotherapy. *Medical Physics*, 20(6):1709–1719, 1993.

[74] O. L. Mangasarian. *Nonlinear Programming*. Number 10 in Classics in Applied Mathematics. SIAM, Philadelphia, PA, 1994.

[75] O. L. Mangasarian. Polyhedral boundary projection. *SIAM Journal on Optimization*, 9:1128–1134, 1999.

[76] O. L. Mangasarian and R. R. Meyer. Nonlinear perturbation of linear programs. *SIAM Journal on Control and Optimization*, 17(6):745–752, November 1979.

[77] O. L. Mangasarian and David R. Musicant. Data discrimination via nonlinear generalized support vector machines. In M. C. Ferris, O. L. Mangasarian, and J.-S. Pang, editors, *Complementarity: Applications, Algorithms and Extensions*, pages 233–251. Kluwer Academic Publishers, 2001.

[78] Marcelo Marazzi. Private communication via email, October 2001.

[79] Marcelo Marazzi and Jorge Nocedal. Wedge trust region methods for derivative free optimization. Technical Report OTC 2000/10, Optimization Technology Center, Northwestern University, Evanston, IL, October 2000. Available as compressed PS document from `http://www.ece.nwu.edu/~nocedal/PSfiles/wedge.ps.gz`.

[80] MathWorks, Inc. FMINSEARCH. MATLAB function. Obtained from the MATLAB command 'type fminsearch' (October 2002).

[81] Bruce A. McCarl. Using GAMS data exchange or GDX files.

HTML document, June 2002. http://www.gams.com/mccarl/gdxuseage.htm.

[82] Bruce A. Murtagh and Michael A. Saunders. MINOS 5.5 user's guide. Technical Report SOL 83-20R, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA, July 1998. Available as PDF document from http://www.sbsi-sol-optimize.com/manuals.htm.

[83] Katta G. Murty. *Linear Programming*. John Wiley and Sons, 1983.

[84] A. Niemierko. Optimization of 3D radiation therapy with both physical and biological end points and constraints. *International Journal of Radiation Oncology, Biology and Physics*, 23:99–108, 1992.

[85] O. B. Olesen and N. C. Petersen. A presentation of GAMS for DEA. *Computers and Operations Research*, 23(4):323–339, 1996.

[86] John C. Pezzullo. Logistic regression. HTML document, August 2001. http://members.aol.com/johnp71/logistic.html.

[87] M. J. D. Powell. Private communication via email, November 2001.

[88] M. J. D. Powell. UOBYQA: unconstrained optimization by quadratic approximation. Technical Report DAMTP 2000/NA14 (Revised), Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge, England,

June 2001. Available as compressed PS document from `http://www.damtp.cam.ac/user/na/reports00.html`.

[89] John J. Rousseau and John H. Semple. Notes: Categorical outputs in Data Envelopment Analysis. *Management Science*, 39(3):384–386, March 1993.

[90] Holger Scheel. Data Envelopment Analysis (DEA) page. HTML document. `http://www.wiso.uni-dortmund.de/lsfg/or/scheel/doordea.htm`.

[91] W. Schlegel and A. Mahr, editors. *3D Conformal Radiation Therapy — A Multimedia Introduction to Methods and Techniques*. Springer-Verlag, Berlin, 2001.

[92] Eugene F. Schuster. Incorporating support constraints into nonparametric estimators of densities. *Communications in Statistics — Theory and Methods*, 14(5):1123–1136, 1985.

[93] L. M. Seiford and J. Zhu. Sensitivity analysis of DEA models for simultaneous changes in all the data. *Journal of the Operational Research Society*, 49:1060–1071, 1998.

[94] D. M. Shepard, M. C. Ferris, G. Olivera, and T. R. Mackie. Optimizing the delivery of radiation to cancer patients. *SIAM Review*, 41:721–744, 1999.

[95] Ronald W. Shephard. *Theory of Cost and Production Functions*. Princeton University Press, Princeton, New Jersey, 1970.

[96] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London, 1986. pp. 52–53.

[97] Léopold Simar and Paul W. Wilson. Sensitivity analysis of efficiency scores: How to bootstrap in nonparametric frontier models. *Management Science*, 44(1):49–61, January 1998.

[98] Léopold Simar and Paul W. Wilson. A general methodology for bootstrapping in non-parametric frontier models. *Journal of Applied Statistics*, 27(6):779–802, 2000.

[99] Léopold Simar and Paul W. Wilson. Statistical inference in nonparametric frontier models: The state of the art. *Journal of Productivity Analysis*, 13:49–78, 2000.

[100] R. J. Tempelman and D. Gianola. Genetic analysis of fertility in dairy cattle using negative binomial mixed models. *Journal of Dairy Science*, 82(8):1834–1847, August 1999.

[101] Russell Thompson, P. S. Dharmapala, and Robert M. Thrall. Sensitivity analysis of efficiency measures with applications to Kansas farming and Illinois coal mining. In Abraham Charnes, William Cooper, Arie Y.

Lewin, and Lawrence M. Seiford, editors, *Data Envelopment Analysis: Theory, Methodology and Applications*, chapter 20, pages 393–422. Kluwer Academic Publishers, 1994.

[102] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

[103] L. J. Verhey. Immobilizing and positioning patients for radiotherapy. *Seminars in Radiation Oncology*, 5(2):100–113, 1995.

[104] Grace Wahba. *Spline Models for Observational Data*. SIAM, Philadelphia, PA, 1990.

[105] Grace Wahba, Yi Lin, and Hao Zhang. Generalized approximate cross validation for support vector machines, or, another way to look at margin-like quantities. Technical Report 1006, Department of Statistics, University of Wisconsin, Madison, WI, April 1999.

[106] Grace Wahba, Yuedong Wang, Chong Gu, Ronald Klein, and Barbara Klein. Smooth spline ANOVA for exponential families, with application to the Wisconsin Epidemiological Study of diabetic retinopathy. *Annals of Statistics*, 23(6):1865–1895, December 1995.

[107] John B. Walden. Private communication.

[108] S. Webb. *The Physics of Conformal Radiotherapy: Advances in Technology*. Institute of Physics Publishing Ltd., 1997.

[109] W. H. Wolberg, W. N. Street, and O. L. Mangasarian. Wisconsin diagnosis breast cancer database (WDBC). FTP documents, 1996. `http://www.cs.wisc.edu/~olvi/uwmp/cancer.html`.

[110] J. W. Wong, M. B. Sharpe, D. A. Jaffray, V. R. Kini, J. M Roberson, J. S. Stromberg, and A. A. Martinez. The use of active breathing control (ABC) to reduce margin for breathing motion. *International Journal of Radiation Oncology Biology Physics*, 44(4):911–999, July 1999.

[111] Chuan Wu, Robert Jeraj, Gustavo H. Olivera, and Thomas R. Mackie. Re-optimization in adaptive radiotherapy. *Physics in Medicine and Biology*, 47(17):3181–3195, September 2002.

[112] Dong Xiang and Grace Wahba. Approximate smoothing spline methods for large data sets in the binary case. Technical Report 982, Department of Statistics, University of Wisconsin, Madison, WI, September 1997. Available as postscript document from `ftp://ftp.stat.wisc.edu/pub/wahba/xwbio.ps`.

[113] Di Yan, Frank Vicini, John Wong, and Alvaro Martinez. Adaptive radiation therapy. *Physics in Medicine and Biology*, 42:123–132, 1997.

[114] S. F. Zavgorodni. Treatment planning algorithm corrections accounting

for random setup uncertainties in fractionated stereotactic radiotherapy. *Medical Physics*, 27(4):685–690, April 2000.

[115] Hao Zhang. Private communication.

[116] Hao Zhang. *Nonparametric Variable Selection and Model Building Via Likelihood Basis Pursuit*. PhD thesis, Department of Statistics, University of Wisconsin, Madison, WI, 2002.

[117] Hao Zhang, Grace Wahba, Yi Lin, Meta Voelker, Michael Ferris, Ronald Klein, and Barbara Klein. Variable selection via basis pursuit for non-gaussian data. Technical Report 1042, Department of Statistics, University of Wisconsin, Madison, WI, October 2001. Available as postscript document from `ftp://ftp.stat.wisc.edu/pub/wahba/tr1042`.

[118] Hao Zhang, Grace Wahba, Yi Lin, Meta Voelker, Michael Ferris, Ronald Klein, and Barbara Klein. Variable selection and model building via likelihood basis pursuit. Technical Report 1059, Department of Statistics, University of Wisconsin, Madison, WI, July 2002.